Science Journal of Kanagawa University 18 : 27-36 (2007)

■原　著■

外部仕様のパラドックス ― 問題点と対策の考察 ―

野口健一郎 [1,2]

Paradox of External Specification
― A Consideration on its Problems and the Measures ―

Kenichiro Noguchi[1,2]

[1] Department of Information Science, Faculty of Science, Kanagawa University, Hiratsuka-shi, Kanagawa 259-1293, Japan
[2] To whom correspondence should be addressed. E-mail: noguchi@info.kanagawa-u.ac.jp

Abstract: The external specifications of information systems including software need to be provided in a clear manner to users and others, which would be an essential requirement for the sound advancement of information society. However, to realize it is not easy. In this paper the intrinsic problems of the definition of the external specifications of information systems are examined. I show that, for a usual information system which is not very simple, it is impossible to define its external specification without utilizing the internal structure of the system and that the ultimate definition results in a sort of implementation of the system. I call this characteristic as *the paradox of external specification*. Next I review how the paradox of external specification affects the actual activities in the following four areas – use of information systems, development of them, interoperation in the network environment, and standardization - and general characteristics are presented. Then the paper discusses the measures, which include the evolution and deployment of specification description techniques such as formal description ones, the improvement of the readability of source programs, and the development of external specifications whose semantics are easy to be understood.

Keywords: external specification, specification description method, specification description language, formal semantics

序論

情報化の進展は著しい。情報化社会を可能にしているのは、コンピュータという 20 世紀に生まれた新しい道具である。コンピュータを利用してさまざまな情報システムが作られており、それらはコンピュータ技術の進展に伴い、ますます高度化しつつある。これらのコンピュータやそれを使った情報システムという道具は、これまで人類が発明した他の道具に比べてはるかに複雑である、という特徴を持っている。そして使う人が、コンピュータや情報システムの機能、特性を細部まできちんと把握し、理解することが難しい、という状況になっている。

利用者からみたコンピュータや情報システムの詳細な機能や特性、すなわち外部仕様は、そもそも開発者にとっても、それをきちんと、利用者に分かりやすく書き表すことが難しい、という根本問題がある。したがって、利用者が外部仕様を正確に把握し、理解することは、そもそも困難なことになってしまっている。今後の本格的な情報化社会の健全な発展のために、この状況をいかに改善するかは大きな検討課題であると思われる。

情報化社会ではネットワークの果す役割がどんどん大きくなっている。情報システムがネットワークを介して繋がるとき、繋ぎのための約束事も情報システムの外部仕様の一種である。これが標準化され、公開されて、誰でもそれに合わせた製品を作れるようにすることも、情報化社会の健全な発展のために

重要である。このとき、これをどのような形で提供するか、というのも外部仕様に関わる大きな課題である。

本論文では、外部仕様の記述、提供に関わる問題点と対策について考察する。まず、外部仕様の本質的な特性と、それから生じる問題点を明らかにする。外部仕様を利用者に分かりやすいように、内部の詳細に立ち入らずに記述することは、ある程度複雑な仕様の場合は不可能であることを論証する。また、外部仕様定義は結局その外部仕様を実現するものを作ることに相当することを示す。(外部仕様が外部的に記述できない、さらに外部仕様定義が作りと等価になってしまうという意味で、これを本論文では"外部仕様のパラドックス"と呼ぶ。) 次に、これらが情報システムの利用や開発に実際的にどのような影響を及ぼしているかを見てみる。それらは外部仕様のパラドックスの例証にもなっている。最後に、対策のためにどのような課題の解決が必要かを述べる。

外部仕様の記述法については、ソフトウェア開発の1段階としてのものは種々行われている。その中で、外部仕様記述の問題点を検討したものとして野口(1990)[1]がある。構造設計およびオブジェクト指向設計での仕様記述については詳細なサーベイがWieringa(1998)[2]にあり、外部仕様記述は未完成の領域であるとしている。

本論

外部仕様定義の問題点

外部仕様

(1) 観念としての外部仕様

コンピュータや、それを使った情報処理機器や情報処理システムを、ここでは一般的に情報処理機械と呼ぶことにする。ソフトウェア（プログラム）は、それを実行するコンピュータと組み合わさって、情報処理機械を構成する。情報処理機械の基本モデルは、入力情報を受け付けて、それを処理し、結果として出力情報を出す、というものである（図1)。

情報処理機械は、それを利用する側（人間または他の情報処理機械）とは入力および出力を介して繋がっている。すなわち、利用する側からは入力情報および出力情報だけが見え、内部の処理は見えない。

図1． 情報処理機械のモデル.

利用する側からみた情報処理機械の特性は、入力情報、出力情報およびその関係によって決まる。それが情報処理機械の外部仕様である。すなわち、外部仕様とは、

・利用する側から見える、または認識できるものである。

・その内容は、入力情報、出力情報およびその間の関係である。

・情報処理機械の内部の処理は、外部仕様には、直接には、含まれない。

第1点および第3点を強調するために、「外部」仕様の名が付いている。以上は外部仕様についての基本的な捉え方であるが、次に述べるような、具体的な捉え方になっていない。これは観念としての外部仕様と言える。

(2) 実体としての外部仕様

外部仕様は、情報処理機械の利用に際して、利用する側にそれを伝えて理解してもらうために、また情報処理機械の開発時に、共同開発者にそれを伝えて理解を共有するために、伝達することが必要になる。伝達できるためには、観念としての外部仕様をものに固定する必要がある。すなわち、外部仕様を定義した「もの」が必要になる。この、外部仕様の定義物は、多くの場合記述物である。外部仕様の定義物が、実体としての外部仕様である。前者の例では利用者向けのマニュアル、また後者の例では外部仕様書と呼ばれる文書がそれに相当する。

以下の議論は、実体としての外部仕様は、それを定義した記述物である、として進める。

外部仕様定義の分類

外部仕様定義は、次の二つの要素からなる。

・入力情報および出力情報の形式の定義

・入力情報と出力情報の関係の定義

外部仕様は、入力メッセージを受け取り、出力メッセージを返す、ということから一つの言語とみなすこともできる。そうみなしたとき、前者は言語の構文規則（シンタックス）に相当し、後者は意味規則（セマンティクス）に相当する。

実際に行われている外部仕様定義は、いくつかのタイプに分類される。

(1) タイプ1：構文のみ型

外部仕様として、構文の側面だけを定義し、意味の側面は無視してしまう、ということがけっこう行われている。これはオブジェクトベースないしオブジェクト指向分野でのインタフェース定義の場合に見られる。インタフェースも、それを利用する側との関係で見ると、そのインタフェースを提供している

プログラムやサーバの外部仕様である。Ada†言語における specification 部[3]、Java††言語の interface[4]、CORBA†††IDL (Interface Definition Language)[5]によるインタフェース記述などはその例で、いずれも構文の側面だけを定義している。

(2) タイプ２：構文＋不完全意味型

タイプ２は、構文についてはきちんと定義するが、意味については完全な定義を目指さず、主要な意味だけを記述する、というものである。利用者向けの外部仕様定義書であるマニュアルの場合はこれが普通である。

(3) タイプ３：構文＋意味型

タイプ３は、構文規則だけでなく、意味についてもきちんと定義するものである。情報処理機械の開発時に作成する外部仕様書はこれを目指している。また、異なるベンダ同士で、製品間のネットワーク接続などを図るためのインタフェース仕様書なども、これを目指している。

外部仕様定義の実際的な問題点

実際に行われている外部仕様定義は、タイプ毎にそれぞれ問題点を含んでいる。前に述べた観念としての外部仕様に照らし合わせると、次のような問題点がある。

(1) 不完全性

タイプ１およびタイプ２は、意味規則が全く、または一部欠けており、外部仕様定義として完全でない。

(2) 実現性

タイプ３について、全ての意味規則を漏れなく記述するのは、大変難しいことが多い。簡単な情報処理機械の場合は問題ないが、実用となるような情報処理機械は通常外部仕様も複雑になり、それをうまく記述する方法が見付からないことが多い。

(3) 表現の内部性

タイプ３について、外部仕様は本来利用する側から見える外部的なものであるはずだが、意味規則を完全に外部的に記述することは難しい、という問題点もある。詳細な意味規則を記述するために、どうしても内部の動きを記述に利用せざるをえなくなることが多い。

意味記述の本質的な問題－記述手法

本来の外部仕様定義であるタイプ３においてもなお、意味定義について実現性と表現の内部性の問題点があることを述べたが、これらは外部仕様の意味規則を記述することに関わる本質的な問題から由来する。まずオートマトン理論[6]をベースとして、基本的な記述手法について考察する。

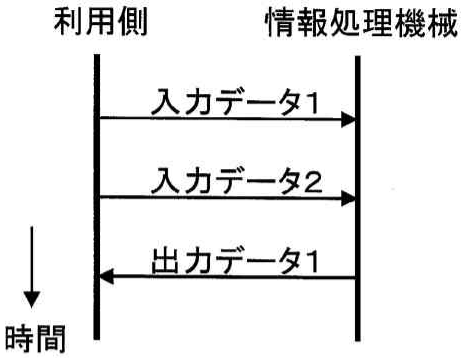

図2. 時系列図（シーケンス図）の例.

(1) 入出力の関係の並べ挙げ

入力情報と出力情報の関係の記述が意味の記述である。これを直接的に行う方法は、関係のインスタンスを並べ挙げることである。その実例が時系列図（シーケンス図）である。時系列図では情報処理機械とその利用側をそれぞれ一本の縦棒で表現し、利用側が与える入力データまたは入力データのシーケンスに対応して、どのような出力データまたは出力データのシーケンスが結果として得られるかを、方向性を持った矢印と一緒に、時系列的に記述する（図２）。

この記述法の利点は、情報処理機械の内部にはまったく立ち入ることなく（時系列図では一本の縦棒で表されるだけである）、利用側から見える入力情報と出力情報だけを使って、入力情報と出力情報の関係が記述されることである。すなわち、まったく外部的な記述になっている。ただし、この方法の限界は、入力情報と出力情報の関係が有限の組合せしかないとき（すなわち関係のインスタンスが有限個のとき）だけ、完全な記述となることである。通常の情報処理機械は入力情報と出力情報の関係は無限の組合せを持っている。そういう情報処理機械に対しても入出力の関係の並べ挙げによる記述が行われることも多いが、それは有限個の例を記述しただけの不完全なものとなる。時系列図は単純な通信プロトコルを表現する手法として広く使われてきているが、ソフトウェアの設計手法としても使われており[1]、UML でも手法の一つとして採用されている[7]。

(2) 内部状態を利用した記述

情報処理機械の入出力の関係に無限の組合せがある場合でも、入出力の関係の規則は有限におさまる。この規則性を記述すればよい。規則性を記述するために、必然的に規則を表現するための媒介の記号類—それは入力情報や出力情報とは別のもの—が必要になる。

† Ada は米国政府の登録商標である。
†† Java は Sun Microsystems、 Inc。の商標である。
††† CORBA は Object Management Group、Inc。の商標である。

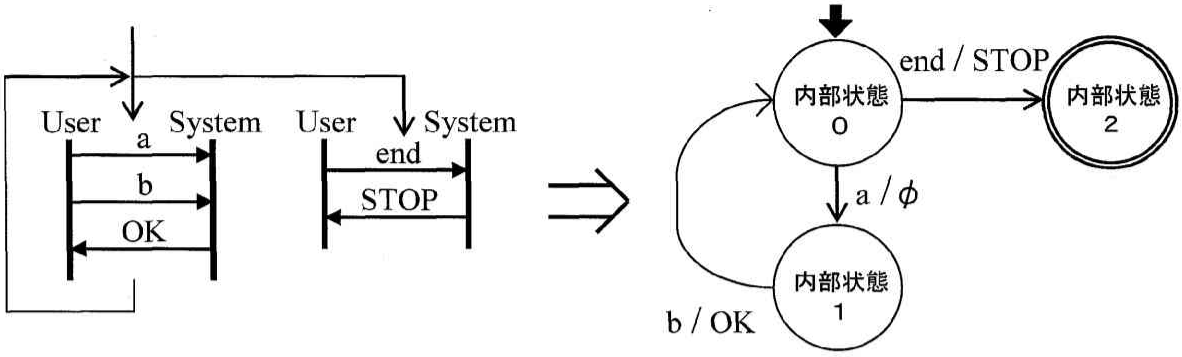

図3. 繰返し型シーケンスの状態遷移図による記述の例.

入出力の関係に無限の組合せがあるもののうちで最も簡単なモデルが有限状態機械である。これは内部状態の概念を導入することにより、それを媒介として入出力の関係の規則性を記述するものである。有限状態機械は、入出力の関係に、正規表現で表すことが可能な、繰返し型の規則性があるものであり、内部状態を用いた記述はその表現法である（図 3)。入出力の関係の規則性は有限状態機械の振舞い、動作とみることもできる。実際の記述方法としては状態遷移図や状態遷移表がある。

有限状態機械モデルに基づいた記述法の利点は、内部状態（それは機械の記憶をモデル化したものである）という概念は利用するものの、それ以外は、利用側から見える入力情報と出力情報のみを用いて、入出力の関係の規則、すなわち意味規則が記述できることである。さらに、内部状態は機械の記憶をモデル化したものであり、機械を利用する場合にそれを認識することが必要なものであるので、有限状態機械モデルに基づいた記述はかなり外部的な記述と言えよう。ただしこの方法の限界は、情報処理機械が有限個の内部状態を持つものである場合だけ、完全な記述となることである。情報処理機械には、構造的に無限の記憶を持つものとして取り扱うべきものや、たとえ内部状態数は有限でも、その数が多すぎて、状態遷移図や状態遷移表では実際的に表しきれないものも多い。そのような場合でも、主要な外部仕様の記述に状態遷移図や状態遷移表が用いられることがある。状態遷移図や状態遷移表は通信プロトコルを表現する手法として、広く使われてきている。またソフトウェアの設計手法としても用いられており、ソフトウェアの状態遷移の表現法は、ISO/IEC 標準ともなっている [8]。階層的な表現などを可能にした拡張手法もある [9]。

なお、内部状態数が多いとき、情報処理機械を複数の有限状態機械に分割できれば、内部状態数の総和はずっと少なくなり、より分かりやすい記述となる。ただし、分割された有限状態機械間での情報のやり取りが出てくる。分割された有限状態機械の記述には、この内部の入出力（それは利用する側からは直接見えないもの）も用いる必要がある [1]。このため記述は外部的なものとは言えなくなる。

(3) 内部記憶および内部記号を用いた記述

情報処理機械が無限の記憶を持つ場合には、有限状態機械より複雑なモデルを適用する必要がある。情報処理機械がプッシュダウンオートマトンとしてモデル化できる場合には、有限個の内部状態と無限長のプッシュダウンスタックを用いて、機械の振舞い、すなわち入出力の関係の規則を記述することができる。情報処理機械の最も一般的なモデルはチューリングマシンになる。この場合には有限個の内部状態の他に、無限長のテープと、内部的な記号（入力情報や出力情報には含まれないもの）も利用して、機械の振舞いを記述することになる。外部仕様の意味記述が、利用する側からは直接見えない内部の記憶装置や内部的な記号を用いた、内部的な記述となってしまう。

(4) 実際の記述手法

実用的な情報処理機械は、内部状態数が多すぎて状態遷移図や状態遷移表では表しきれなかったり、または無限の内部状態を持つとして取り扱う必要のあるものが多い。実際の記述手法はそのような一般的な情報処理機械の仕様を記述できなければならない。

状態遷移技法を拡張して、機械の内部変数を導入し、状態遷移に伴う機械の動作に内部変数に対する操作を許すことによって意味記述をすることが行われている。これは内部状態の一部を内部変数で表現することに相当し、“拡張状態遷移図”または“拡張状態遷移表”と呼ばれる。状態遷移技法とプログラミング言語的な表現を組み合わせることにより、記述の自由度がずっと広がる。この手法は OS の仕様記述用としてかなり以前に提案された [10]。通信分野では OSI（開放型システム間相互接続）[11]の通信プロトコルの意味記述にも用いられている。Estelle はそれを記述言語化したものである [12]。この手法、

あるいはその変形は、ソフトウェア設計手法において、外部仕様記述の手法として用いられている[1)13)]。特に構造設計およびオブジェクト指向設計において多く採用されており[2)]、UML でもこの一種が採用されている[7)]。

仕様記述言語を用いて、形式的に仕様を記述することが行われている。VDM-SL (Vienna Development Method - Specification Language)[14)]、Z 言語[15)]、B[16)]などは集合論や述語論理に基づく数学的記法を採用している。しかしこれも内部の変数などを利用した、内部的な記述になっている。述語論理を用いた記述は、プログラミング言語による処理型の記述とは異なるものの、結局は外部仕様を実現したインタープリタとみることができる。(プログラミング言語の意味定義は結局その言語のインタープリタである、という見方は schmidt(1996)[17)] にある。)

ソフトウェアの場合、ソースプログラムは、完全な意味の記述も含んだものである。すなわち、プログラミング言語そのものが意味記述の手法のひとつである。

意味記述の実際的な手法は、結局外部仕様を何らかの方法で実現したインタープリタを作ることに相当し、記述は内部的なものになる。

(5) 外部仕様記述の本質的な問題点のまとめ

以上で述べたモデル化および記述手法の比較を表 1 に示す。複雑な情報処理機械では、意味の規則を記述するために、入力情報と出力情報に加えて、内部的な情報を補助として用い、それらの間の関係を記述することになる。

問題点をまとめると、次のようになる：

法則 1 ：情報処理機械は、有限状態機械としてモデル化できるものより複雑なものの場合は、その完全な外部仕様を、内部の振舞いに立ち入らずに、外部的に記述することはできない。

すなわち、観念としての外部仕様は、実体としては存在しない（存在できない）ことが多い。

意味記述のもう一つの本質的な問題－定義の循環論

(1) 外部仕様定義の循環論

情報処理機械の外部仕様を完全に、曖昧さ無く記述するためには、記述に用いる手法が、外部仕様記述を受け取り理解する人にとっても、明確で曖昧さの無いものでなければならない。

単純な記述手法(たとえば時系列図や状態遷移図、状態遷移表）の場合はこの問題はない。しかし、仕様記述にもっと複雑な記述言語を用いる場合は、まずその記述言語自身の定義をきちんと行う必要が出てくる。すでに述べたように、言語と外部仕様は本質的には同じものであり、言語の定義は（その言語を受け付ける情報処理機械の）外部仕様の定義である。すなわち、ここに定義の循環論の問題が生じる：

法則 2 ：情報処理機械の外部仕様定義を完全に、曖昧さ無く行うために、ある記述言語を用いると、次にその記述言語の定義の問題が生じる。これは循環論になり、各レベルの記述言語として定義が必要なものを用いる限り、記述による外部仕様定義は完結しない。

次にこの循環論を断ち切る方策を考察する。

(2) 定義済み記述言語の利用

外部仕様定義をある記述言語を用いて行い、またその記述言語の定義を別の記述言語で行う、という場合、あるレベルの記述言語が定義済みのものであれば、それより上のレベルの記述言語の定義も明確になり、もとの外部仕様定義も明確なものとなる。

定義済み記述言語としては、単純な記述手法であって定義の問題が無いもの、数学の記法、あるいはその地域の人なら誰でも理解できるはずである自然語などが用いられる。単純な記述手法や数学の記法の場合も、実はその定義は自然語で行われているはずで、結局自然語が最終的な定義済み言語となっている。

表 1. 情報処理機械のモデルと意味記述の方法

情報処理機械のモデル（複雑さのレベル）	意味記述として何の関係を記述するか				記述手法の例
	外部入力 外部出力	内部状態	内部入力 内部出力 (内部要素間)	内部記憶・ 内部変数	
入出力の関係が有限個	✔				時系列図
有限状態機械	✔	✔			状態遷移図（表）
	✔	✔	✔		複数の状態遷移図（表）
無限状態	✔	✔		✔	チューリングマシンの記述
一般（制約なし）	✔	✔	✔	✔	仕様記述言語 拡張状態遷移図（表） プログラミング言語 （ソースプログラム）

最終的な定義済み言語が自然語である、ということは、自然語の曖昧さの問題が伴う可能性がある。厳密に考えれば、自然語はその文法がはっきり分かっているわけでなく、また完全な辞書があるわけでもない。外部仕様や言語の定義に向いた言葉でもない。したがって、外部仕様定義を記述によって行うというのは、完全な定義にならない可能性がある。

法則３：情報処理機械の外部仕様は、明確で、曖昧さの無い記述手法で記述できる場合を除いて、完全に、曖昧さなく記述することは困難である。

すなわち、ある程度複雑な情報処理機械については、実体としての外部仕様も、存在が難しいことになる。

(3) 定義機械による定義

外部仕様定義の循環論は、記述のアプローチだけでは断ち切ることが難しいことを述べた。これを断ち切る本質的な方法がある。それは、あるレベルの記述言語に対して、その定義を記述で与える変わりに、その言語を受け付け、理解する機械を作って、与えることである。そのような機械をその言語の定義機械と呼ぶ。定義機械により、その言語が明確に、曖昧さなく定義される。意味、すなわち入出力の関係も、定義機械の上で試すことにより、明確に知ることができる。

ある言語の定義機械とは、その言語のインタープリタとして機能する情報処理機械である。すなわち、その言語で記述されたもの（それは一つ上のレベルの言語の定義）を入力して、記述のとおりに動作する。すなわち、その言語で記述された一つ上の言語の定義と、その言語の定義機械が組み合わさって、一つ上の定義機械が構成されることになる。結局、もとの情報処理機械の外部仕様定義についても、それを定義する機械、すなわち記述対象の外部仕様を持つ情報処理機械が構成されることになる。この状況を図４に示す。図は、情報処理過程を形式的に表す手法を用いて[18)19)]、言語とその定義機械は同じ記号で表現し、また、記述物は記述対象の外部仕様や言語と記述言語をそれぞれ分数の分子、分母として表記している。図では、言語Ｌに定義機械Ｌを与えることにより、それより上のレベルの言語および外部仕様についてもそれぞれ定義機械が構成されることを示している。以上をまとめる：

法則４：情報処理機械の外部仕様を、明確で、曖昧さ無く与える殆ど唯一の方法は、その外部仕様を持つ機械を作って、与えることである。

外部仕様のパラドックス

外部仕様は、観念としてのものも存在が難しく、また実体としてのものも存在が難しいこと、さらに完全な外部仕様定義はそれを実現する機械を作って与えることにより達成されること、すなわち外部仕様が結局は作り（実装）と等価であることまでが論証された。外部仕様が外部仕様で無くなってしまった。これ（法則１から４まで）を“外部仕様のパラドックス”と名付ける。

外部仕様のパラドックスの影響

外部仕様のパラドックスが複雑な情報処理機械、特にソフトウェアの利用や開発にどのような影響を及ぼしているか、を見てみる。なお、これらは外部仕様のパラドックスの例証でもある。

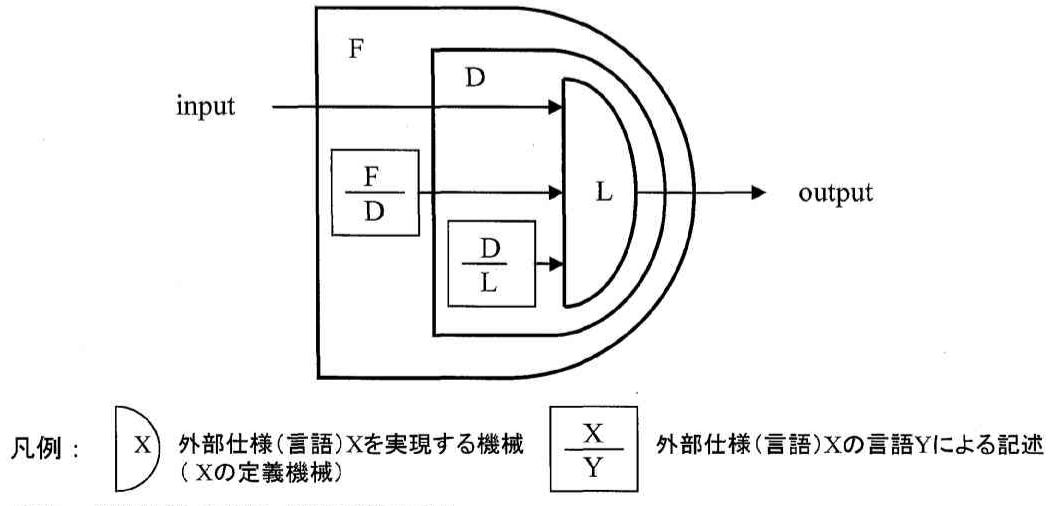

図4. 外部仕様（言語）定義の階層の例.

情報システム利用への影響

補則１：ソフトウェアの利用者には完全な外部仕様情報は提供されない。

外部仕様を完全に、詳細に記述することが困難であることから、こうならざるをえない。実際上も、多くの利用者向けマニュアルは主要な外部仕様のみを記述している。これは、詳細な仕様まで記述しようとすると、マニュアルが厚くなりすぎ、多くの人にかえって読みにくくなってしまう、というのも理由の一つである。なお、マニュアルでは外部仕様の意味規則は自然語で書かれることが多く、その点からも曖昧さがあり、不充分である。

補則２：利用者が詳細な外部仕様を知る最後の手段は、実際に製品を動かしてみることである。

実は、外部仕様のパラドックスから、これは本質的な方法である。利用者は、試行によって、詳細な入出力の関係を知ることができる。なお、構文規則がマニュアルなどに完全に記述されていれば、試行は容易になる。また、仕様そのものが類推が付きやすく作られていれば、それによっても試行が楽になる。ただし、試行で分かるのは入出力の個別の関係（インスタンス）であり、それから規則性を帰納するのは、一般には容易ではない。それは一種のリバースエンジニアリングであり、時間のかかる、発見的なプロセスとなる。

情報システム開発への影響

補則３：ソフトウェアの内部の設計に着手する前に、外部仕様をすべて明確に規定するという理想は、実用的にも理論的にも満たされない。内部の設計を進めることにより、外部仕様も明確にすることができる。

少し複雑な製品の場合、外部仕様の記述は内部的にならざるをえない。すなわち、内部の構造や内部の変数を前提にすることにより、外部仕様の記述もできる。すなわち、外部仕様の記述のために、なんらかの形で内部設計を行うことが必要になる。

これは、外部仕様の設計と内部処理の設計において、ウォータフォール・モデルに従った開発はできないことを意味する。

補則４：ソフトウェア開発において、開発がかなり進むまで、何を開発しているかが完全には分からない。

補則３を別の形で述べたものである。

また、補則３ないしは４は、開発管理の面でも、かなり深刻な影響を及ぼしている。

補則５：ソフトウェアの開発計画（予定の日程、予定のコストなど）を立てる時には、開発の対象（何を開発するか）は完全には定義されていない。

開発対象物は、外部仕様によって定義される。しかし、それは開発が進まないとはっきりしない。開発計画を立てる時は、開発対象物がどういうものになるか、よく分かっていない。結果として、製品の開発計画は実際とはずれたものになってしまう。その結果としての日程の遅延やコストの増大、いうことも多い。

開発を、契約により請け負わせる・請け負うときにも同様の問題が生じる。

補則６：ソフトウェアの開発契約締結時に、契約の対象（何を開発するか）が完全には定義されていない。

開発契約の発注仕様書には、何を開発するかをきちんと書くことができない。したがって、契約する日程や費用も、妥当なものかどうか分からない。開発契約に関わるトラブルのもとにもなる。

開発の過程で、外部仕様が完全なものになったとき、それの文書化（外部仕様書としての）の問題がある。これには、最後の解決策がある。

補則７：外部仕様の詳細な記述（意味も含めて）は可能である。完成したソースプログラムがそれである。

法則３と４との関係では、ソースプログラムは明確で、曖昧さの無い記述であり、またその外部仕様を持つ機械を実現しているものである。なお、外部仕様の完全な形式定義も、一種のソースプログラムと見ることもできる。

ネットワーク環境での相互運用の実現への影響

ネットワーク環境で、製品間の相互運用を実現することが、今後極めて重要になっていく。このためには、製品同士が同じ通信プロトコルを理解し、それを用いて対話できなければならない。通信プロトコルも、それを実現する製品の外部仕様である。ただし、特定製品に所属する外部仕様というより、複数製品が共有する外部仕様である。インターネット時代においては、それは規格として標準化され、誰でもがアクセスでき、かつそれを実装する製品を作れることが望まれる。このためには、通信プロトコル規格が文書化されることが必要である。ただしこれ

は必要条件ではあるが、十分条件ではない。

補則８：文書化された通信プロトコル仕様だけに基づいて、製品間の相互運用を実現することは難しい。

通信プロトコルは、状態遷移モデルで記述できるレベルの基本的な通信プロトコルについては問題ないが、アプリケーションプロトコルには複雑なものが多い。それらを文書として、詳細かつ完全に記述することは難しい。しかし、プロトコルが詳細かつ完全に記述されていないと、それに基づいて実装した製品間の相互運用は達成できない。

OSI は、標準規格文書をまず作り、それに基づき製品開発を行い、製品間の相互接続、相互運用を達成しようという国際的なプロジェクトであった。適合性試験や相互運用性試験の必要性も認識されて、そのための努力もなされたが、全体としては文書中心のアプローチであったことが、成功できなかった大きな要因といえる。

補則９：通信プロトコルを実装したソースコードを公開かつ権利放棄して、誰にでも利用可能にすることは、相互運用性を実現する確実な方法である。

インターネットの基本プロトコルである TCP/IP は UNIX[†] の BSD 版と共に広まった[20]。また、Sun Microsystems の NFS (Network File System)[21]もソースコードの流通により広まった。

なお、公開だけではだめか、権利放棄まで必要か、ということについては次のようになる。たしかに、ソースコードが公開されれば、そこから外部仕様を読み取ることができる。しかし、実体としての外部仕様は記述物であり、たとえば著作権で守られているソースコードを読み取って、独立の著作物を作ることは困難である。IBM 社のネットワークアーキテクチャである SNA は、ソースコードレベルの詳細なプロトコルがマニュアルとして公開されている（プログラムと有限状態機械を組み合わせた記述手法が用いられている）[22]が、著作権を放棄することはしなかった（1984 年の EC との独禁法裁判の和解の結果[23]）。

補則 10：通信プロトコルの参照用実装を利用して相互運用のテストを行うことにより、文書化された通信プロトコル仕様の完全でない部分を補うことができる。

参照用実装（reference implementation）とは、その外部仕様を正しく実装した原器として利用可能にされるものである。なお、先行開発され、市場に普及したデファクト製品が、実質的な参照用実装の役割を果すこともある。

仕様の標準化への影響

通信プロトコル以外の分野での、仕様の標準化への影響を述べる。

補則 11：文書化された標準仕様に基づいて独立に開発された製品間では、緩やかな互換性が実現される。

通信プロトコル以外の分野では、ハードウェア仕様の差などを考慮して、標準規格といえども、製品間の完全な互換性を要求するものではなかった、という経緯もある。プログラミング言語や OS の分野では、仕様の細部には実装依存とされている部分がある。ただし、ネットワーク環境での使用を考慮したプログラミング言語では、事情は変わってきている。

対策と今後の課題

外部仕様のパラドックスを認識しつつ、２１世紀の情報システムの外部仕様の提供方法をどのようにすべきかは、重要な検討事項である。

外部仕様記述の提供

今後の情報化社会においては、情報システムの詳細な機能、特性を、利用者が知ろうと思えば分かるようにすべきであると考える。しかし、外部仕様のパラドックスから、実用的な、したがってある程度複雑な情報システムやソフトウェアの厳密な外部仕様の記述は、内部的にならざるをえない。

数学的ないしは論理学的手法を用いた形式的な仕様記述手法、オートマトン理論・形式言語理論と実際的なプログラミング手法を組み合わせた手法などは単に開発手法としてだけでなく、より広い仕様提供手法として使えよう。これらの進化と普及を期待する。なおこれらの手法は、記述手法自身の仕様定義の問題を解決するためにも、定義機械により実行可能なものであることが望ましい。

利用者には、主要な外部仕様だけを知ればよい人から、詳細な外部仕様を必要とする人までいろいろなレベルがある。外部仕様記述も、主要な外部仕様の記述から、詳細な記述まで、階層的に構成できる手法の研究も意味があると思われる。

[†] UNIX は The Open Group の登録商標である。

ソースプログラムの利用

ソフトウェアの場合、ソースプログラムを完全な意味記述を含んだ文書として利用することは有効な方法であり、プログラミング言語の進歩に伴い、より現実的な方法になってくると思われる。たとえばオ

・ソースプログラムの読解性の向上

ソースプログラムを文書としても取り扱う場合、その読解性の問題がある。一つの解決策はより論理的な記述を可能にすることで、構造化プログラミング言語、オブジェクト指向言語などはその方向での進化でもある。読解性についてはさらに、プログラミング言語仕様の国際化・地域化の問題がある。現在のプログラミング言語は英語ベースのものが多く、英語圏以外の人々にとっては、読むのに骨が折れる。それぞれの地域の言語をベースにしたプログラミング言語の表現を可能にすること（自国語プログラミング）[24,25]が、その解決策になる。ソースプログラムの各国語表現への変換は、コンピュータにとっては容易な処理である。

・知的財産との関係の解決

ソフトウェアのソースプログラムには外部仕様だけでなく、実装の技術もすべて含まれる。ソフトウェア開発を業とする企業にとっては、最も重要な知的財産である。その公開には、この関係の解決が必要である。社会的なセキュリティを考えた場合、情報化社会を実現するためのソフトウェアが完全にブラックスボックスでよいのか、という問題もある。広く一般に公開しないまでも、特定の条件でソースプログラムを閲覧可能にする措置が必要になると考えられる。

ソースプログラムを広く利用可能にして、プログラム開発の促進と、信頼性の確保を図ろうという動き（Open Source Software）も出てきている[26]。情報化社会のインフラとなるソフトウェアは、社会財としてこのカテゴリになってくる可能性もある。

外部仕様の作り方

外部仕様の意味定義の問題を軽減するために、外部仕様そのものの作り方も検討してゆく必要がある

・他言語で意味を定義済みの構文要素の利用

外部仕様で用いる構文要素（語や記号）に、他の言語（特に自然語）で意味を定義済みの構文要素を借りるようにすると、新たな定義をしなくても、意味が理解できる。

・外部仕様の国際化・地域化

意味を定義済みの構文要素として、その地域の言語ベースの構文要素を利用すれば、さらに意味理解が容易になる。GUI(Graphical User Interface)の分野はかなりそうなっている（ただし、英語をカタカナ語にしたものが多い問題はある）が、他の外部仕様も、それを考えていくべきである。なお、外部仕様を国際化・地域化する際には、異なる地域表現の間でも相互運用性を確保するようにすべきである。それはコンピュータにとっては難しいことではない。

ブジェクト指向でクラスを継承するとき、親クラスの振舞いはソースプログラムを読むことによって完全に理解される。ソースプログラムを利用する場合の課題を述べる。

標準仕様・共通仕様の提供方法

情報システムは、通常改版が行われることにより、新しい製品が生み出される。そして、情報システムの外部仕様は、改版が行われても将来的に引き継いでゆかれる共通的な仕様と、特定の版の個別の仕様に分かれる。前者の共通仕様は、製品開発上明確に管理する必要がある。また、利用者に対して、またはその情報システムと相互運用する別のシステムのために、将来への約束として明確に示す必要がある。ソースプログラムは仕様の完全な定義として有効ではあるが、共通仕様と個別仕様を区別してくれない。共通的な外部仕様を意味まで含めて明確に記述し提供する方法は課題である。

システムがネットワークによって相互に接続される場合、製品間にわたる標準的・共通的なインタフェース仕様に従い、その範囲だけを使って接続が実現される。したがって、個々の製品が改版され、別製品になっても、継続して接続が行える。このように、標準規格化されている外部仕様、インタフェース仕様は、標準的・共通的な範囲を取り決めている。それを実装した製品の外部仕様とは同じではない（後者は前者を包含しているが）。標準規格化される外部仕様、インタフェース仕様の記述と提供の方法も、同様に課題である。

なお、アーキテクチャという用語がソフトウェアの世界でも使われることが多くなった。この用語の本来の意味は、体系化された標準インタフェースの集合、というものである。標準仕様・共通仕様の記述と提供方法の課題とは、アーキテクチャの記述と提供方法の課題とも言える。

結論

情報システムの外部仕様が、明確な形で利用者等に提供されることは、情報化社会が健全に発展していくための要件であると思われるが、その実現は必ずしも容易ではない。

本論文では、まず外部仕様定義の本質的な問題点を考察し、ある程度複雑な仕様の場合は、

・外部仕様は外部的に記述できない、

・外部仕様定義は作りと等価になる

など、外部仕様がその名前と矛盾する特性を持つことを論証し、４つの法則として示した。そしてそれを外部仕様のパラドックスと名付けた。次に、外部仕様のパラドックスが実際面でどのように影響しているかを、情報システムの利用、開発、ネットワーク相互運用、および仕様標準化の４つの局面について考察し、11個の補則としてまとめた。これらは外部仕様のパラドックスの例証にもなっている。最後に対策および今後の課題について考察した。外部仕様のパラドックスが存在するため、外部仕様の容易な提供方法はない。

・仕様の形式記述手法などの進化と普及

・ソースプログラムの読解性の向上

・意味理解が容易な外部仕様の作成

などが解決策であり、また今後の課題である。

文献

1) 野口健一郎 (1990) *ソフトウェアの論理的設計法*, 共立出版, 東京.
2) Wieringa R (1998) A Survey of structured and object-oriented software specification methods and techniques. *ACM Comput. Surv.* 30: 459-527.
3) American National Standards Institute, Inc. ANSI/MIL-STD-1815A-1983 (1983) *The Programming Language Ada Reference Manual*, Lecture Notes in Computer Science 155, Springer-Verlag, New York.
4) Gosling J, Joy B, Steele G and Bracha G [村上雅章訳](2006): *The Java TM 言語仕様 第3版*, ピアソン・エデュケーション, Tokyo.
5) Object Management Group, Inc. (2004) *Common Object Request Broker: Core Specification, Version 3.*
6) Hopcroft JE, Motwani R and Ullman JD [野崎昭弘他訳]：オートマトン 言語理論 計算論Ⅰ・Ⅱ第２版, サイエンス社 (2003).
7) Rumbaugh J, Jacobson I and Booth G (1999) *The Unified Modeling Language Reference Manual.* Addison-Wesley, Reading, Mass.
8) ISO/IEC 11411:1995 (1995) *Information Technology - Representation for Human Communication of State Transition of Software*, International Standard.
9) Harel D (1988) On visual formalism. *Comm. ACM* 31: 514-530.
10) 野口健一郎, 元岡達 (1973) オペレーティング・システムの記述に関する一考察. *情報処理* 14: 98-105.
11) ISO 7498:1984 (1984) *Information processing systems - Open Systems Interconnection - Basic Reference Model.* International Standard .
12) ISO 9074:1989 (1989) *Information processing systems - Open Systems Interconnection - Estelle: A formal description technique based on extended state transition model*, International Standard.
13) 渡辺政彦 (1998) *拡張階層化状態遷移表設計手法 Ver.2.0.* 東銀座出版社.
14) ISO/IEC 13817-1:1996 (1996) *Information technology - Programming languages, their environments and system software interfaces - Vienna Development Method - Specification Language - Part 1: Base Language.* International Standard .
15) Potter B, Sinclair J and Till D [田中武二監訳](1993) *ソフトウェア仕様記述の先進技法—Z言語.* プレンティスホールトッパン, 東京
16) Wordsworth JB (1996) *Software Engineering with B.* Addison-Wesley, Harlow, England.
17) Schmidt DA (1996) Programming language semantics. *ACM Comput. Surv.* 28: 265-267.
18) Sklansky J, Finkelstein M and Russel EC (1998) A formalism for program translation. *ACM.* 15: 165-175.
19) 野口健一郎, 元岡達 (1969) 情報処理過程の形式的表現. *昭和 44 年度電子通信学会全国大会予稿集.* p. 883.
20) Leffler SJ, McKusick MK, Karels MJ and Quarterman JS [中村明, 相田仁, 計宇生, 小池汎平訳] (1991) *UNIX 4.3 BSD の設計と実装.* 丸善株式会社, 東京.
21) Sun Microsystems, Inc. (1989) *NFS: Network File System Protocol Specification. Network Working Group Request for Comments: 1094.*
22) IBM Manual SC30-3112-2 (1980) *Systems Network Architecture Format and Protocol Reference Manual: Architectural Logic*, Third Edition.
23) IBM [日程コンピュータ訳] (1984) IBM-EC 独禁法問題和解：IBM 提出の確約書全訳, *日経コンピュータ* 10月1日号: pp. 191-198.
24) Noguchi K (2006) An internationalization approach to native language based programming. *Proc. 4th Int. Conf. Computer Science and Its Applications.* pp. 87-91.
25) 鈴木康彦, 野口健一郎, 後藤英一 (1999)：Java をターゲットにした自国語プログラミングの実験. *情報処理学会 第58回全国大会予稿集.* 5M-8.
26) Raymond ES [山形浩生訳] (1999) *伽藍とバザール オープンソース・ソフト Linux マニフェスト.* 光芒社, 東京.