社団法人　情報処理学会　研究報告
IPSJ SIG Technical Reports

2005－ＮＬ－168（17）
2005／7／23

PIIからマニュアルへの参照を作成する方法について

加藤 直孝† 有澤 誠‡

†(株)日本アイ・ビー・エム ナショナル・ランゲージ・サポート 〒242-8502 神奈川県大和市下鶴間 1623-14
‡慶應義塾大学 環境情報学部 大学院 政策・メディア研究科 〒252-8520 神奈川県藤沢市遠藤 5322
E-mail: † katosan@jp.ibm.com, ‡ arith@sfc.keio.ac.jp

あらまし 現在多くのソフトウェアプログラム(以下プログラムと略す)は，ユーザーとのインターラクションを前提としており，ユーザーとの意思疎通のために文字列を用いる．この文字列を PII(Program Integrated Information)と呼ぶ．この文字列はプログラムから分離したテキストファイル上に存在する．プログラムをローカライズするには，この文字列を各国語に翻訳する．一方，プログラムのマニュアルは本文中に PII 文字列を引用している．また GUI の図にも PII 文字列を含む．ところが，PII の更新があると，関連するマニュアルの箇所も更新する必要がある．マニュアル上の更新箇所を発見するには，従来はグローバル検索機能(Grep 等)により検索し，該当する PII かどうかを調べていた．この方法では検索後 PII と関係のない文字列を検討する必要があり，大きな障害になっていた．本研究では，PII 文字列からマニュアルの PII 文字列へ参照リスト(ハイパーリンク)を作成する方法を開発し実装した．本手法により，多数の PII 文字列以外の文字列を調べる必要はなくなり，また文字列ごとに検索が完了するのを待つ必要もなくなった．本稿は最後に問題の考察をとおして，Propositional Logic との対比で PII とプログラムの意味(Semantics)に関する新しい理解を提示する．

キーワード 翻訳，マニュアル，シナリオ，文字の外部化，国際化，地域化，PII，言語，意味，命題論理，ロジック

A method of creating reference to PII strings in manuals

Naotaka KATO† Makoto ARISAWA‡

† Translation Service Center, IBM Japan, Ltd. 1623-14 ShimoTsuruMa, Yamato-shi, KANAGAWA, 242-8502 Japan
‡ Faculty of Environmental Information and Graduate School of Media and Governance, Keio University 5322 Endo, Fujisawa-shi, KANAGAWA, 252-8520 Japan
E-mail: † katosan@jp.ibm.com, ‡ arith@sfc.keio.ac.jp

Abstract Most of the software programs require interaction with users. Such programs use text strings to communicate with users. Those strings are separated from its program into text files and are called "Program Integrated Information (PII)." Those strings in text files are translated into each language when the program is localized. The manuals of the program include those strings to refer to the strings shown in the program's GUI. When there is an update to PII, the affected parts of the manuals need to be updated as well. The affected strings in the manuals are searched by global search functions such as Grep. But the search picks up a lot of non-PII strings in manuals and those non-PII strings needs to be removed from update candidates by human inspection. Translators often give up searching if the number of strings is too large for inspection. Our research found a method to create a reference list (hyperlinks) of PII strings in text files to the PII strings in manuals that do not include non-PII strings. This method does not require translators to check non-PII nor to wait for Grep completion. Lastly, but not least, this paper presents a new view to the relation between PII and program semantics by referring the propositional logic.

Keyword translation, PII, Program Integrated Information, manual, scenario, meaning, model, GUI, propositional logic

1. はじめに

プログラムを国際化するには，プログラム中のストリング(文字列)を何カ国語かに翻訳する．一般に，この作業はプログラムの開発研究所では行わず，翻訳の業務に特化した組織が行う．開発研究所では翻訳する可能性のある文字列は別のテキストリソースファイルに分離する．このテキストリソースファイルにはキーおよび分離したストリングを置く．そして，プログラム中にキーを埋め込み，キーの参照により適切な文字列を得る．IBM の場合，プログラムのマニュアルも PII テキストリソースファイルを翻訳する組織と同じ組織が翻訳を行なっている．

通常マニュアルはプログラムの操作方法を説明する．その操作説明に GUI 上の文字列を引用する．あるいは GUI イメージをマニュアル内に持ち，そのイメージ内の文字列を

参照する．GUI 内のほとんどの文字列は PII になっているので PII とマニュアルは密接に関係する．たとえば PII を変更したり，PII の翻訳を修正すると，その変更に合わせてマニュアルも変更する必要がある．GUI イメージの再キャプチャーと本文中の PII 文字列の修正である．ところが，PII の修正があってもマニュアル上の該当箇所の発見は困難である．本稿はこの困難に対処する方法を提示する．

PII とマニュアルとの関係は，前稿[1]の「PII の概念構築」で「PII の連鎖」として述べた．また前稿[2]では，GUI 上の文字列が PII テキストファイル上のどの文字列と対応するかを知る手法を説明した．本稿はテキストファイル上の PII 文字列からマニュアル本文中の文字列への参照リストを作成する手法を述べる．

本稿は実装した手法とは別に，理想的なマニュアル開発環境を述べるとともに，「PII の概念」という観点から考察を加える．その考察を通して，プログラムに意味を与えるには PII の ID としてのキー(key)と文字列(string)の2つの要素からなる集合が必須であることを説明する

2. 本研究の背景

2.1. 具体的な問題と参照の有用性

PII やマニュアルの翻訳では新規翻訳の作業より，バージョンやリリースアップの作業が大半を占めている．リリースアップ等で翻訳した PII の文字列を修正すると，プログラム中の PII の表示とマニュアルの内容に食い違いが生じる．この場合，マニュアル中の PII 文字列を修正する必要がある．この修正はマニュアルを翻訳する部署がマニュアル更新時に修正箇所を発見し修正する．ところが PII の文字列でマニュアルを Grep すると，PII の文字列と関係のない文字列を多数表示する．1つの PII の翻訳修正に，マニュアル上の多数の文字列を1つひとつ確認し該当箇所を探すことになる．マニュアル翻訳期間が短いとマニュアル上の PII の文字列の修正そのものをあきらめざるを得ない．たとえば，Dassault 社の CAD System プログラムでは，全部の PII の個数は 17 万個を超え，原典のマニュアルは約 100 冊存在し，日本語に翻訳しているものは約 50 冊ある．ひとつの PII 文字列でマニュアルを Grep すると，通常数十個の文字列をリストする．このリストから該当する PII 文字列を発見することは容易ではない．

PII 文字列のマニュアルへの参照リストは，PII の修正以外にも役立つ．PII の翻訳では PII の文脈(使用シナリオ)が分らないことが重大な問題になっている．PII 翻訳時や PII 検証時に PII を使用しているマニュアルの箇所が分れば，PII を使用する文脈が分る．PII 文字列のマニュアル上への参照リストを活用すれば PII の文脈が分り，翻訳の質の向上が期待できる．

2.2. PII の連鎖

上記2．1の問題は前稿[1]で説明した「PII の連鎖」を容易に保てないことに起因する．図1に本来の連鎖の様子を

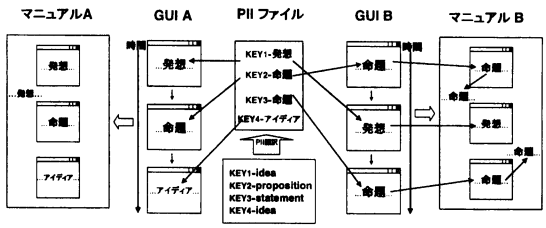

図 1　PII の連鎖

示す．

ここでは key3 の statement を「命題」と翻訳している．この命題は GUI B でも「...命題...」と出現する．マニュアルは GUI をイメージとして持ち，そこには「...命題...」とある．そのイメージを参照するマニュアル本文中にも「...命題...」が出現する．通常この連鎖とはまったく関係のない「命題」という言葉が本文中に多数出現する．そこでたとえば，statement の翻訳を「ステートメント」に修正したとすると，マニュアル上のどの「...命題...」を修正すればよいかが容易には判断できない．

2.3. PII の連鎖の失敗

図2に PII の修正で「PII の連鎖」の崩れた例を示す．ここでは，PII の翻訳を「基準」から「標準」へ修正している．プログラムの GUI は正しく PII を出力する．しかし，マニュアル上では，修正以前の「基準」という PII の文字列が残っている．本来は，マニュアル上の文字列で key1 に該当する部分は「基準」から「標準」へ修正する必要がある．ところがマニュアルを参照しても，「基準」という言葉が key1 の「基準」かどうかは分らない．

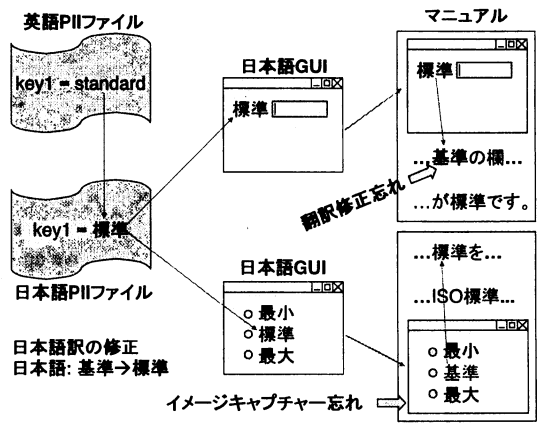

図 2　PII の連鎖の失敗

マニュアル上で，PIIは次の2つの形態で出現する．

(1) マニュアル本文中の文字列として

(2) GUIをキャプチャーしたイメージファイルとして

よって，マニュアル上の PII を修正するには，本文中の文

字列の修正と，GUI のキャプチャーの取り直しによる修正が必要である．

本稿で述べる参照は，PII からマニュアル本文中への参照である．マニュアルは通常，修正を必要とする本文中の文字列の近くに，同じ文字列を含んだ GUI のイメージが存在する．そのため，本文中の PII 文字列の場所がわかれば，GUI イメージも修正可能になるケースが多い．本稿の参照リストは GUI イメージの質向上にも役立つ．

このことに関連して，GUI イメージから OCR で文字列が読み取れないか調査してみた．その結果は GUI イメージの解像度はキャプチャーしている細かな文字に対して非常に低く(あらく)，英語についてもまったく実用に耐えないと判断した．

2.4. 連鎖発見が困難である理由

2.4.1. ID（Identification）の重要性

PII の連鎖は PII の ID に対して成立している．ところが，GUI にもマニュアルにも ID を参照する仕組みがない．PII における ID とは，PII のフォルダー名，ファイル名，キー名，およびバージョン等の世代情報である．前稿[2]では，GUI 上でその ID を突き止めることを可能にした．そのため，GUI から PII への連鎖を発見することはできるようになった．ところが，逆方向の PII から GUI への連鎖を知ることは困難である．PII からマニュアルへの連鎖の発見も同様である．

この方向についての問題はソフトウェアとシナリオの関係に起因する．シナリオが決まらないとソフトウェアの動作を一意に決定できないからである．

2.4.2. シナリオの不可逆性

「PII の連鎖」の失敗を発見することが困難な理由は「シナリオの不可逆性」による．マニュアルには操作シナリオが書かれている．プログラムの操作シナリオがわかれば，プログラムを操作して，GUI を出力できる．そして，前稿[2]で紹介した PII の TVT 用の ID を活用すれば，PII ファイル中に対応する文字列を見つけることができる(図 3 参照)．

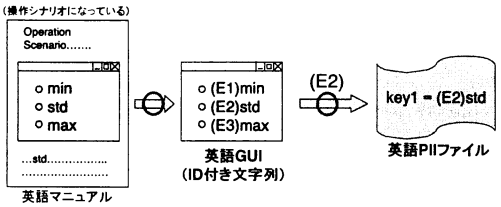

図 3 マニュアルから PII ファイル上の文字列への参照

ところが逆に，PII ファイル中の文字列から，GUI 上の文字列を指摘することはできない．プログラムを操作し GUI を出力する以外に GUI を見る方法はない．GUI がわからないので，マニュアルのどこにその GUI を掲載しているかもわからない(図 4 参照)．

シナリオから PII は特定できるけれども，PII からシナリオは

図 4 PII ファイル上の文字列からマニュアルへの参照

特定できない．この事実を「PII に関するシナリオの不可逆性」と命名する．

2.5. 理想的なマニュアル開発環境

理想としては次のような形になることが望ましい．GUI のイメージをキャプチャーすると，Bit Map 情報と同時に PII の文字列および ID を自動的に取得できるようにする．イメージファイルは著作権情報その他とともに，これらの PII に関する情報も持つようにする．このようなイメージファイルをマニュアル上におく．

また，マニュアルの本文中で使用する PII 文字列にも同様に文書中に PII の ID を埋め込み，参照できるようにする．この PII の ID は PII の Identity Feature（通常は PII ファイルのフォルダー名，ファイル名およびキー名）としての ID を持ち，何国語なのかといった言語や世代(バージョンやリリース)の情報も含める．GUI のイメージのファイル中には PII の文字列のデータと PII の ID のデータを含める．本文中の PII 文字列にも，その文字列のプロパティーとして同様のデータを含める．具体的には，HTML で<pii>タグを定義し，適切なプロパティーを定義する．将来，プログラムがマニュアルを自動生成した場合 ID を付けることは容易であり，マニュアル上の PII 文字列の部分的な修正は自動で可能になる．

ただし，理想的な環境が整うまで何年も待つことはできない．そこで本稿では次章で，現状において実行可能であり，従来の方法よりはるかに効率的で有効な手法を述べる．

3. PII からマニュアル中の文字列への参照作成

2つの新しい方法を実装した．第1の方法は HTML のタグを活用しない方法で，第2の方法は HTML のタグを活用する方法である．第1の方法も無駄ではない．しかし，第2のタグを活用する方法がはるかに効率的で効果が大きい．本稿では第2の方法を詳しく説明する．

3.1. HTML のタグを活用しない方法

第1の方法は HTML のタグを活用しない方法である．HTML ファイルのフォルダー群から PII の文字列で HTML ファイルを検索する．そして，PII 文字列ごとに対象となる HTML ファイルをリストする．この方法を CAD System Software である Dassault 社の CATIA V5 に対して実装した．全 PII(約 170000 個)に関して全マニュアル(約 100 冊)を対象に検索すると，PC で約 30 日の運転を必要とする．この方法は，PII やマニュアルを限定すれば活用できる．たとえ

ば，修正した PII 文字列リストに限定した検索をする．従来の Grep で参照先を求めるよりはこの方法は効率が上がる．けれども，飛躍的な改善はもたらさない．

3.2. HTML のタグを活用する方法

第2の方法は，HTML 作成時に GUI 上の文字列であることがわかるようにタグを使う方法である．具体的には，<kbd>タグで GUI 上の文字列を囲む．この方法は参照先を PII の文字列に限定でき，リストが実用可能な範囲になる．CATIA の全 PII に対する全マニュアルを処理するのに要する時間は 10 分以下になり，第1の方法よりもずっと小さい．次節でこの第2の HTML のタグを活用する方法を CATIA V5 における実施例をもとに説明する．なお CATIA V5 は CSS(Cascading Style Sheet)を用いて画面上の文字列を識別可能にするため，従来から<kbd>タグは挿入済みである．<kbd>タグの活用はこの点においても重要である．

3.3. HTML のタグを活用する方法の実施例

リンク元のテキストファイルとリンク先の HTML ファイルを次の手順で準備する．準備する PII のテキストファイルおよび HTML のマニュアルは，ともに本来の PII のファイルやマニュアルからコピーして作業を行なう．次のマニュアル側の準備(1)以外はプログラムで実行する．プログラムには Perl を使用した．

＜マニュアル側の準備＞

(1) 原典の HTML フォーマットのマニュアル制作時にコンピュータ画面上の文字列(PII 文字列)は<kbd>...</kbd>タグで囲んでおく．(この作業はマニュアル作成の規則である．)

(2) 完成したマニュアルに対し，<kbd>タグの直前にアンカータグ<A NAME=...>を挿入する．アンカータグの NAME は<kbd>タグに対する ID にする．たとえば，<kbd>タグが html のテキストファイルの 53 行目の 1 番目のタグなら kbd-000053-01 という ID にする．

＜PII ファイル側の準備＞

(1) 全 PII を一定の規則に従ったフォーマットで，1 つのテキストファイルにし，リストを作成する．

(2) 上記 PII リストから順に PII を取り出す．取り出した PII の文字列に等しい<kbd>タグで囲まれたマニュアル上の文字列を検索する．等しい文字列があれば，その<kbd>タグの直前のアンカータグをジャンプ先にしたハイパーリンクを作成する．リンク情報を上記(1)の PII リスト上の情報とともに最終的な結果となるテキストファイルに追加していく．マニュアル上に<kbd>で囲まれた同じ文字列が複数あればリンク情報を複数書き込む．ただし，「Yes」といった頻繁に出てくるリストは量が多すぎてテキストファイル参照時の障害になるので，たとえば 50 箇所以上参照先のあるものは参照せずに，特別に扱う．なお，HTML 文字列検索は，マニュアル全体の検索用中間リストテーブルを作成し検索処理に要する時間を少なくする．これには ID として HTML ファイルに挿入したアンカータグが重要な役割を果たす．CATIA の場合，このテーブルにより，タグを活用しない検索の約 5 千倍の処理速度が実現できる．

最後に，準備した HTML ファイル群を HTML サーバー上に置く．最終的な結果となるテキストファイルは，実際に PII 文字列を参照するユーザーがテキストエディターで開き，特定の PII 文字列から HTML ファイルにジャンプして，マニュアルの内容を確かめる．

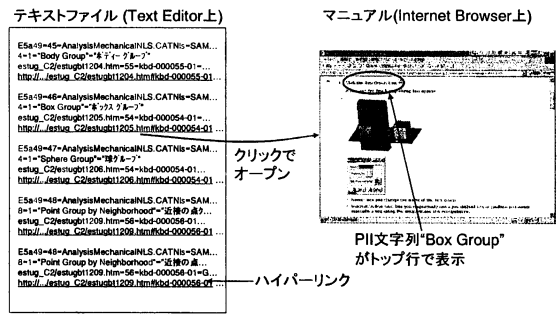

図 5 PII からマニュアルへの新しい参照方法

最終的には図5のようなテキストファイルを生成する．図5の左図はテキストエディターでファイルを開いたところである．同じパラグラフ構成を繰り返している．パラグラフ間は空白行で分かれており，1つのパラグラフに1つの PII になっている．

3.4. 実際に使用したフォーマット

PII テキストファイルフォーマットを次に示す．等号で分離した文字列フォーマットを使用した．ここで「...」は省略を示す．下線表示のあるところがハイパーリンクである．

E5a49=46=AnalysisMechanicalNLS.CATNls=SAM...
4=1="Box Group"="ﾎﾞｯｸｽｸﾞﾙｰﾌﾟ""
estug_C2/estugbt1205.htm=54=kbd-000054-01=...
http://.../estug_C2/estugbt1205.htm#kbd-000054-01

1行目: E5a49 は PII のための TVT (Translation Verification Test) ID (前稿[2])で，英語 Release15, GM, ファイル番号 49, 46 行目，ファイル名，key 名，と続く．

2行目: 4 は PII 全体で 4 個のキーで同じ文字列「Box Group」を定義している．1 はマニュアル上で「Box Group」とあるのは 1 箇所のみであることを示す．英語の文字列，日本語訳の文字列と続く．1 行目の ID がここでのハイパーリンク先(4行目)と1対1対応で対応しているかどうかわからない．そこで，TVT の ID を表示するプログラムで，マニュアルのシ

ナリオを実行し確認する.

3行目: 左から順に, HTMLファイルのフォルダー名とファイル名, <kbd>タグの出てくるファイル中の行番号, <kbd>タグのID. <kbd>タグのIDの最後の01はその行の中の1番目の<kbd>タグを意味する.

本研究の実装では, 2行目の英語の"Box Group"で英語のマニュアルに対して検索を実行している.

4行目: マニュアルへのハイパーリンクである. 実際に使用する時には, http サーバーのアドレスに合わせて, エディターで上位のアドレスを全部置換する.

4. 実施例での特徴的なデータ

CATIA Version 5 におけるPIIファイル上のPII文字列に対するマニュアル上のPII文字列の比率は, 表1のようになる. 最後の比率はPIIファイルとマニュアルとの関係を知りうるPIIファイル比率である.

表 1 PII文字列からマニュアルへのリンク比率

PIIファイル中の文字列の分類	比率
PII文字列の総数に対し, マニュアル上に同じPII文字列が存在するものの比率	24%
PII文字列の総数に対し, PIIファイル上にその文字列が1個のみ存在するものの比率	2%
PIIファイルの総数に対し, そのファイルの要素であるいずれかのPII文字列がマニュアル上のPII文字列と一致するファイルの比率	67%

次に, マニュアル上のPIIの文字列で出現頻度の多いものをリストする. 明らかにドキュメントのCAD Systemのマニュアルの特徴が出ている.

表 2 マニュアル上のPII文字列出現頻度

文字列	回数	文字列	回数
OK	4201	Apply	613
Tools	1584	Edit	560
Options	1103	Properties	524

表2に続いて, Start, General, Insert, View, File, Open, Close, Preview, Save, Infrastracture, Mechanical Design, Name, Options..., Cancel, Drafting, New, More, Add, Type, Remove, Show,...と続く. いずれも100回以上である. 翻訳は簡単そうに見える. ところが「Name」とするか「名前」とするかは簡単でない. たいてい「Name」と「名前」の翻訳は混在する. PII IDなくして修正は極めて困難である.

5. タグを活用する方法の特徴と利点

タグを活用する方法は, 次の3つの特徴と利点を持つ.

(ア) キーボードタグ<kbd>の活用の利点

(1) <kbd>タグでマニュアルの対象文字列をPIIの文字列に限定できる.

(2) <kbd>タグの使用は「PIIの連鎖」の発見に有効であると同時に, CSS(Cascading Style Sheet)を用いて, ユーザーにGUI中の文字列であることを喚起できる利点がある.

(3) <code>タグや<sampl>タグはPIIとはあまり関係しない文字列にも使われる可能性が高い. しかし<kbd>タグはその可能性が低い. 実際にはHTML作成の規則とし, <kbd>タグをGUI上の文字に限定して使用している.

(4) すでに完成しているHTMLマニュアルにも適応が可能である. 重要なマニュアルからのみ適用していくことも可能である.

(イ) アンカータグ<A>の挿入の利点

(1) アンカータグの挿入により, テキストファイルから該当箇所へのハイパーリンクが可能になる. アンカータグの名前(NAME=ID)を利用しマニュアル上のPII文字列の場所にIDを振ったことになる.

(2) アンカータグを活用し, 中間リストテーブルを作成し, 検索処理時間が短くなる. 中間リストテーブルは, マニュアルのHTMLフォルダー, ファイル名, および上記アンカータグのIDと<kbd>タグ内の文字列で構成する. 各PII文字列はこの中間リストテーブルに対し検索処理を行なうので, タグを活用しなと約30日かかる処理が, 10分以下で完了する.

(ウ) 単純なテキストフォーマットによるリストの利点

(1) PIIファイル上のキー1つひとつに対し参照先をあらかじめリストとして準備している. そのためGrep等による待ち時間がなく, 全体の状況を把握しやすい. たとえば次のような状況が一目でわかる.

- 全PIIキー中にいくつ同じ文字列があるか.
- マニュアル上にはいくつ対象となる文字列があるか.
- マニュアル上で同じ文字列は同じファイル中に集中しているか, いろいろなファイルやフォルダーに分散しているか.

(2) テキストエディターからハイパーリンクで, マニュアルの該当箇所にリンクできるので, マニュアル上の文字列を表示する作業効率が高く, 生産性を高める.

(3) 特殊なプログラムを用意しなくても, テキストエディター(e.g. K2 Editor[4])の機能が活用できる. 全PIIをひとつの単純なテキストデータにまとめリンクを作成しているので, 柔軟性が高く, Editorの文字検索(含む正規表現検索)や, Grep, ハイパーリンク等の機能を自由に活用できる.

(4) PIIからの参照データの準備にはPerlの環境が必要である. しかし, データを活用するユーザーは特別な環境を必要としない.

(5) 結果リストの仕様の変更がきわめて容易である.

6. 「PII の概念」による考察

PII の ID は PII のフォルダー名，ファイル名，キー名，世代情報に相当する．この章ではなぜ PII の ID にマニュアルを埋め込む必要があるかを考察する．

文字列単独でも，PII の ID 単独でもプログラムに意味を与えることはできない．PII の ID に文字列を結び付けるときに意味が生まれる．従ってプログラムに「意味」を与えるには「文字列」の情報だけではなく，「PII の ID」の情報も必要である．

6.1. PII ID の背後のロジック

マニュアル上に PII の ID 情報がなければ，PII の修正は困難である．PII 上のどの文字列がマニュアル上のどの文字列と対応しているかがわからないからである．現状ではマニュアル上に ID がないので，マニュアル上のシナリオを流して，GUI 上の PII と 1 対 1 対応を取りながら，修正を行なう．もちろん GUI 上では，前稿[2]で述べた TVT 用の ID を文字列に挿入している必要がある．もし TVT 用の ID を挿入しなければ，シナリオに従ってプログラムのロジックを追う必要がある．これは ID を突き止める一種のデバッグである.それゆえ，ID が背後にロジックを持っていると考えてもよい．

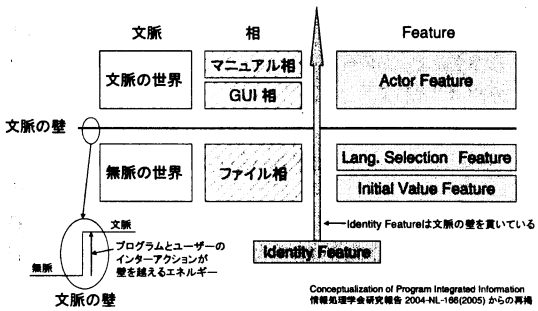

図 6 PII の概念：文脈，相，および PII Features

ここで「PII の概念」の図を前稿[1]から再掲する．PII にはファイル相，GUI 相，マニュアル相が存在する．ファイル相に対応するのが「無脈の世界」，GUI 相とマニュアル相に対応するのが「文脈の世界」である．Feature の観点からは，Actor Feature, Identity Feature, Initial Value Feature, Language Selection Feature が存在する．ID が背後にロジックを持っていると言うことを，前稿[1]の「PII の概念」のなかで理解すると次のようになる．マニュアル相，GUI 相，ファイル相に出現する文字列は ID を文字列に置き換えたものと考えてよい．置き換えの Feature が Initial Value Feature であり，Language Selection Feature である．その ID は Actor Feature によってコントロールされている．Actor Feature とは結局プログラムのロジックとプログラムのユーザーとのインターラクションの結果生まれる．従って，ID は背後にロジックを持つことになる．ロジックを動作するには通常クロックが必要である．従って Actor Feature には時間の流れがある．Actor Feature が時間の力を借りて文脈を生み出している．

6.2. 意味のユニットと意味の分離

PII を Propositional Logic と Predicate Logic の対比で述べると，PII の世界は基本的に Proposition で記述することを要求している．主語や目的語や動詞を分離して PII の連結で文章を作ろうとすると，翻訳で破綻をきたすからである．主語の PII，動詞の PII，目的語の PII と並んだ英語の文章を日本語にはできない．PII が出現する順番はプログラムのロジックが制御しているからである．PII はひとつの意味のユニットである必要がる．

国際化を必要とするプログラムは文字列を外部のテキストファイルに分離し，プログラム中には代わりにキーを置いた．この分離はプログラムの「ロジック」と「自然言語」の分離をもたらした．「自然言語」の分離は実際には「意味」の分離をもたらしている．

6.3. Propositional Logic との対比

Propositional Logic における「意味（Semantics）」は，atom をある世界の Proposition に結びつける(associate)ところにある．結びつきがないと意味はないけれど，結びつくと意味が生まれる．そして，atom と proposition の結びつきを interpretation とよぶ．

ここで，この結びつきを「=:=」で表現すると次のように書ける．

atom =:= proposition

この式は「atom に proposition を結びつける」と読む．Propositional Logic に「意味」を与えるにはこの2つの集合が必要となる．atom だけでは「意味」を成さず，また proposition だけでも「意味」を成さない．

PII の話に戻ると，Initial Value Feature や Language Selection Feature で行なっていることは，文字列(string)をキー(key)に結びつけることである．この結びつきを「=:=」で表現すると次のようになる．

key =:= string

これは，key が atom に相当し，string が proposition に相当する．6 章の冒頭で述べたように key だけでは「意味」を成さず，また string だけでも「意味」を成さない．プログラムに意味を与えるには key と string の2つの要素からなる集合が必要である．ここで key とは PII の ID を指す．key はフォルダー名等も含む．通常，この key =:= string の結びつきをプログラムに実装するには，key = string の代入式を用いている．

Propositional Logic では atom に結びつける要素は何らかの世界のドメインの中にある．string の場合だと，英語のドメイン，日本語のドメイン，フランス語のドメインというようにドメインを変更できる．たとえば，PII の用語のドメインを各国の言語から独立したドメインにし，そのドメインが NL_meaning_ID をそれぞれに持つと仮定すると，次の要素からなる集合も意味を持つ．

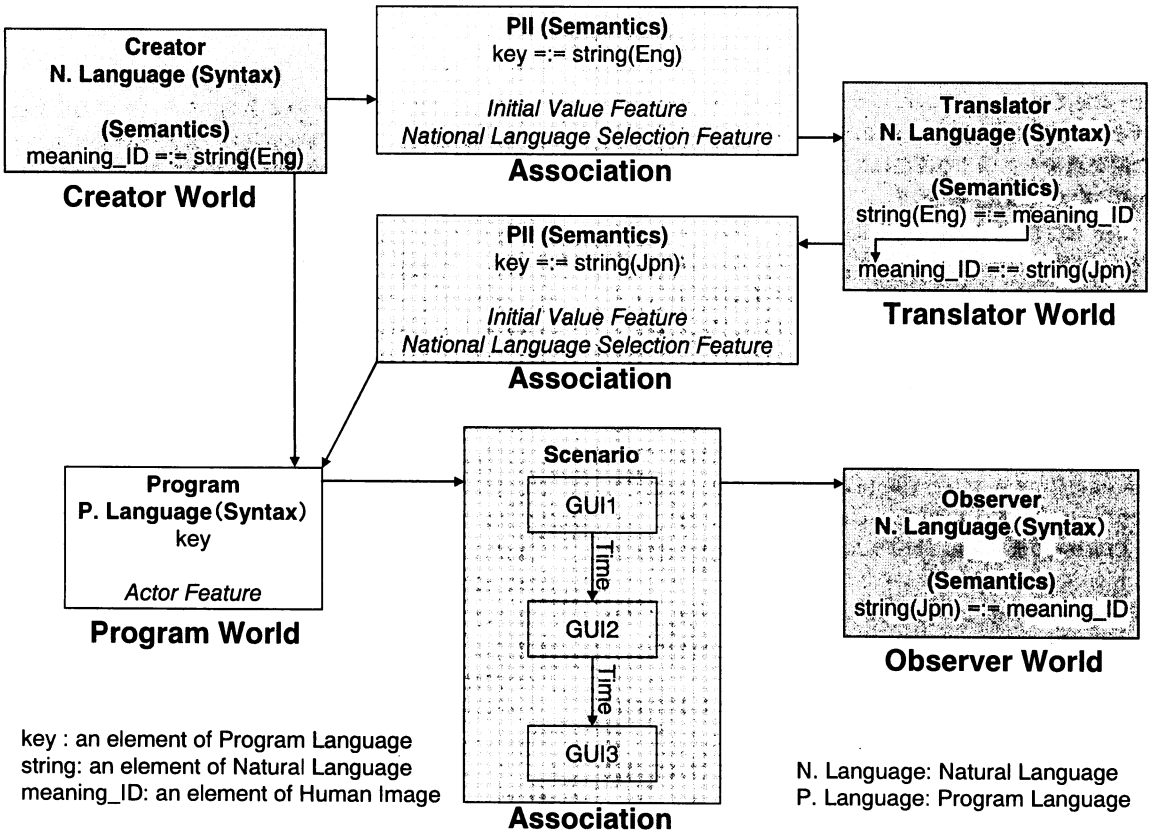

図 7　プログラム世界と人間世界の結びつき(Association)

key =:= NL_meaning_ID

ここでも，key 単独でも，NL_meaning_ID 単独でも意味をなさない，この2つの要素を結び付けないとプログラムに意味は生まれない．

key はプログラムの世界の要素であり，NL_meaning_ID は人間世界の要素である．自分の世界の言葉で相手の世界を理解し，相手を活用する．結びつくと(Associate)人間にとっても，プログラムにとっても世界が広がる．その結果、両者は互いに意味ある存在になる．次節では，図を用いてこれらの関係をより詳しく説明する．

6.4. プログラム世界と人間世界の結びつき

図7はプログラムの作者(Creator)，PII の翻訳者(Translator)，プログラムの観察者(Observer)を濃い陰影のボックスで示す．プログラムは陰影のないボックスで示す．縦の列で真ん中の列の半分陰影のあるボックスはそれぞれ PII ファイル(PII(Semantics))とシナリオ(Scenario)を示す．濃い陰影の部分はいずれも人間の世界であり，陰影のない部分はプログラムの世界である．そして，半分陰影のある部分がプログラム世界と人間世界の結びつく(Associate)ところであり，プログラム言語と自然言語が結びつくところでもある．そして，キー(key)と文字列(string)との結びつき(Association)によりプログラムに意味が生まれると同時に文字列にも意味が生まれる．なお，矢印は情報の流れを示す．図7の矢印ではそのため Observer からプログラムへの入力情報は省略している．そのため図7ではユーザー(User)とはしないで，観察者(Observer)とした．関係の深い「PII の概念」の「Feature」はボックス内にイタリック文字で示している．

Creator はプログラムと PII の両方を作成する．翻訳者(Translator)は string(Eng)の文字列を meaning_ID と理解し，日本語の string(Jpn)に翻訳する．プログラムには key が存在し，key を GUI 上へ出力するときには PII のデータを参照して string(Jpn)を出力する．プログラムの観察者は string(Jpn)を見て，meaning_ID を意味すると理解する．Observer と Program の間には GUI が存在し，時の流れとともにシナリオができる．プログラムのシナリオ(Scenario)はプログラム世界と人間世界のコミュニケーションの軌跡である．ところがそのシナリオも PII が存在しなければ意味を持つことはない．逆にシナリオが意味を持つにはキー(key)と文字列

(string)との結びつき(Association)が必要である．そして，マニュアルにはシナリオを含む．key =:= string の string に変更があった場合，それにともなうシナリオ上(マニュアル上)の変更場所を知るには，シナリオ上での key の出現箇所を知る以外に方法はない．2．5節でマニュアルの理想的な開発環境を主張する根拠はここにある．

6.5. プログラムの語彙

key はプログラムの語彙(Vocabulary)である．世界が異なれば，語彙は異なる．たとえば，同じ文字列が6個の異なる動きをするkeyと結びつくとする．プログラムの世界は6個の語彙を持つ．ところが人間の文字列はそれぞれに対応する語彙を持たないのでそのような結びつきになる．

PIIファイルはkeyに原典の英語を結びつけた時点でプログラム言語の世界から英語の世界に翻訳を行なっている．この翻訳はプログラムの作者(図7 Creator)が行なっており，key に何らかの独立した意味(meaning_ID)を思い浮かべており，その meaning_ID を string(Eng)と翻訳している．

Creator の世界ではプログラム全体の意味は次のようになる．

ProgramA: {(key1 =:= meaning_ID1), (key2 =:= meaning_ID2), (key3 =:= Meaning_ID3)..., (keyN =:= Meaning_IDN)}

人間がプログラムの意味を決めるのに必ずしも文字列は必要ない．意味の ID との集合の集合で決めうる．各 Meaning_ID には別途 string の候補がある．各国の言語によりこの string の候補の集合は切り替わる．

Meaning_ID1 の候補: {string1, string2, ..., stringN}

Meaning_ID2 の候補: ...

プログラムの原典作者や翻訳者はこの候補から string を選ぶ．前稿[1]の「3．1．1 PII の作者と観察者」では作者(Creator)と観察者について述べた．上記の表現は作者の立場での表現である．観察者(Observer)の立場では次のような表現になる．

Program A: {(key1 =:= string1), (key2 =:= string2), (key3 =:= string3)..., (keyN =:= stringN)}

string1 の候補:{meaning_ID1, meaning_ID2, ..., meaning_IDN}

string2 の候補: ...

meaning_ID を導入すると，作者と観察者の違いは，meaning_ID, string(文字列)の順番の違いに帰結する．

6.6. GUI 上の文字列の原典はプログラムの語彙

プログラムの語彙は本来 key である．その key を人間の世界に結びつける役割を果たすのが PII ファイルの対の集合である．key と文字列の集合からなる PII ファイルはプログラム言語から自然言語への翻訳ファイルと言ってよい．

マニュアル上の操作シナリオの原典は，本来は英語ではなく key で構成した GUI である．その原典の key を失った状態(現状のマニュアルの状態)で key と対になる文字列を修正することは本来困難である．

7. おわりに

本稿では PII ファイル上の文字列からその文字列に対応したマニュアル上の文字列を発見し，参照リストを作成する方法を提示した．本稿で説明した方法を用いることにより，PII の修正にともなって影響を受けるマニュアル上の文字列を発見しやすくなる．それによりバージョンアップにともなうマニュアルの品質低下の防止が容易になった．また，従来文脈が分からずよい翻訳ができなかった PII 文字列に対し，マニュアル上で文脈を発見しやすくなった．同様に，PII の翻訳品質の向上が期待できる．

本稿ではさらに理想的なマニュアル開発環境を述べるとともに，PII の概念を通してなぜそのような環境が必要かを述べた．またその考察を通して，プログラムは PII のキー(key)と文字列の集合の集合で意味が決まること示した．PII ファイルはプログラム言語と自然言語の接点に存在し，プログラム言語にあるいは自然言語に意味を与える重要な存在であることを示した．

意味の問題で重要なのは，主体が何かということである．通常は人間である．自然言語の意味を解き明かしたいならば，価値判断の主体をプログラム言語に与えてはどうだろう．プログラム言語が自然言語を活用し意味(価値)を見出す．コンピュータが人間を活用し、人間に意味を見出す．

謝辞

本研究で作成したテキストファイルは 100M byte を超えるものもあった．それにもかかわらず，フリーソフトウェアの K2 Editor[4]は基本的には問題なく使用できた．1 テキストファイルが 400M byte のものですら動作した．この場を借りて K2 Editor の作者関係者に感謝の意を表したい．

文　献

[1] 加藤直孝，有澤誠，“Conceptualization of Program Integrated Information”，情報処理学会研究報告，2004-NL-166, pp. 39-46, 2005

[2] 加藤直孝，有澤誠，“翻訳検証テストのためのストリングリソース ID”，情報処理学会研究報告，2004-NL-162, pp. 95-102, 2004

[3] 加藤直孝，有澤誠，“Program Integrated Information (PII)翻訳のための PII 概念構築”，言語処理学会第11回年次大会論文集，pp. 771-774, March, 2005

[4] K2 Editor, K2 Software's page, http://k2top.jpn.org/

[5] Nils J. Nilsson, Artificial Intelligence: A New Synthesis, Morgan Kaufmann, 1998

[6] Zohar Manna, Richard Waldinger, The Deductive Foundations of Computer Programming, Addison-Wesley, 1993

[7] Zohar Manna, Mathematical Theory of Computation, Dover Publications, 2003

[8] IBM, Designing Internationalized Products, National Language Design Guide Volume1, Fifth Edition, February 2004