2004－NL－162　(14)
2004／7／15

翻訳検証テストのためのストリングリソース ID

加藤 直孝[1]　　有澤 誠[2]
(株)日本アイ・ビー・エム ナショナル・ランゲージ・サポート[1]
慶應義塾大学 環境情報学部[2]
katosan@jp.ibm.com

概要: プログラムが表示するメッセージストリング（Program Integrated Information, PII）の翻訳検証（Translation Verification Test, TVT）は、開発研究所と各国の TVT テスターが協力して行う。本研究はプログラムが表示するメッセージストリングに対応する外部化したストリングリソースの場所を TVT テスターがすぐに発見できるようにすることを目的とする。そのため TVT 用に新たにストリングリソース ID を導入した。この ID は GUI に表示する外部化した全メッセージストリングの前に付加する。TVT テスターはこのストリングリソース ID から、メッセージストリングのリソース中の場所を特定できる。本稿ではこの ID と ID を振る手法、および TVT における使用方法を説明する。本 ID はテスト対象となるプログラムとは独立に実装でき、翻訳すべきメッセージストリングを外部化するあらゆるプログラムに適用できる。ストリングリソース ID を用いる本手法は IBM が翻訳を担当する France Dassault System 社の CAD/CAM Program である CATIA® の TVT テスト用に実用化し、さまざまな効果を確認した。ストリングの場所を発見する効率は 10 倍以上になった。また本稿では、Java プログラムへの適用例を示すことにより、本手法があらゆるプログラム言語に対し適用可能なことを示す。

キーワード: 翻訳、翻訳検証テスト、リソース ID、文字の外部化、国際化、地域化、PII、Java、リソースファイル、ストリング ID

String Resource ID for Translation Verification Test

Naotaka Kato[1]　　Makoto Arisawa[2]
IBM Japan, Ltd. Translation Service Center.[1]
Keio University　Environmental Information[2]
katosan@jp.ibm.com

Abstract: PII(Program Integrated Information) is a string message that is displayed in GUI. The PII strings are externalized from program itself to text files for translation. TVT(Translation Verification Test) testers and program development team test translation of PII(Program Integrated Information). This paper introduces string resource ID for TVT. This string resource ID is shown in front of each string PII. TVT testers can find the location of a target string in resource files with the help of this resource ID. This paper explains the ID, how it is implemented, and its use in TVT. This ID is implemented independently of the target program, and is applicable to all programming languages that externalize string resources. We applied this approach to CATIA® (CAD/CAM program from Dassault System) PII translation for which IBM is responsible. We confirmed useful features of the ID in the test. Time to find the string location became more than ten times shorter. This paper shows the application of this approach to a Java program as well to validate the universality of this approach.

Keywords: translation, TVT, Translation Verification Test, Resource ID, string externalization, internationalization, localization, Java, resource file, string ID

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必要がある。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っている。開発研究所では翻訳する可能性のあるストリング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ストリングを得る。

外部化したストリングの翻訳検証テストは、プログラムをテストシナリオに沿って操作しながら行う。翻訳したストリングに誤りがあればテキストファイル中のストリングを修正する[1, 2, 3]。従来は表示しているストリングの出所を探すのに、OS やエディターに付属の grep を用いてストリングの場

所を探していた。この方法では、ストリングの数が大きくなると探索に数十秒単位の時間を必要とする。また、ストリングを組み合わせて出力するときや、同一のストリングが複数あるときは grep で場所を特定することはできなかった。

本研究はGUI上に表示しているストリングがどのテキストファイルのどのストリングから出力しているのかを効率よく発見するために行った。そのために、GUI 上の出力ストリングの出所がわかるように、テキストファイル上の各ストリングの前に短い ID を振るプログラムを開発した。テスト対象のプログラムはこの ID を出力ストリングの一部として GUI 上に表示する。たとえば、ストリングが「角度」なら「(36.20)角度」のように GUI 上に ID「(36.20)」を表示する。

開発したプログラムは実際 CATIA Version5 Release13 の日本語 TVT で使用し有効性を確認した。このリソース ID の導入によりリソースファイルのストリングの探索に要する時間が、30 時間から 1 ないし 2 時間に減少した。なお、TVT 全体の時間は 120 時間で、プログラムが ID を表示するように準備する作業は 10 分程度である。

本稿では、GUI 上に表示するストリングの情報を PII(Program Integrated Information)と呼び、PII の翻訳に対するテストは TVT(Translation Verification Test)と呼ぶ。また、外部リソースに分離せずプログラム中に埋め込んでいるストリングを hard code したストリングと呼ぶ。なお、IBM では PII の翻訳に Translation Manager を使用している。本稿ではこれを TM と略す。TM が管理するフォルダーの単位を Information Unit と呼び、これを IU と略す。また、IBM では翻訳業務に特化した組織を TSC(Translation Service Center)と呼ぶ。本稿でも TSC と略す。

本 ID はプログラム開発後の hard code の有無のチェックや PII の翻訳時にも活用できる。ただし、本稿では以下 TVT に焦点を当てて研究の内容を説明する。

2. 本研究の背景

図1にPIIの翻訳および検証の流れを示す。IBMではIUフォルダーの右側および破線(中央よりやや下の横の破線)より下が TSC(Translation Service Center)の仕事となり、TVT は④から⑦の部分の作業となる。

ストリング・リソースファイルを中心に翻訳のプロセスを説明すると、次の①から⑦のような流れになる。図1の①から⑦がそれぞれの説明に対応している。図1の破線より下が従来の手法に追加した本稿で述べる手法部分である。

① 開発研究所はプログラム開発段階で翻訳する可能性のあるストリングリソースはすべて外部化(externalization)する。

② 外部化したストリングリソースは、キー(key)とそれに対応する原語(通常は英語なので、以下英語とする)からなる。このテーブルの形式には、データベースやXMLや平テキスト等がある。ただし、ほとんどのものは平テキスト形式になっている。

たとえば次のようなフォーマットになっている。

key1 = Message String A
key2 = Message String B
key3 = Message String C

キーはメッセージの内容を示す長いストリングであることが多い。また、テキストファイルはフォルダー(ディレクトリー)別に分類していることも多い。プログラム中ではフォルダー名、ファイル名、キー名を指定することにより、実際に出力するストリングを特定する。

③ この外部化したテキストファイル中で翻訳の必要なファイルのみが開発研究所から各国の TSC に渡る。開発研究所と TSC の管理の都合上、テキストファイルあるいはそれを含むフォルダーをさらにグループ化する。IBM はこのグループ化したものを IU(Information Unit)と呼んでいる

④ この IU 単位に分けたキーとストリングのマップテーブルを翻訳サポートソフトウェア上で翻訳する。IBM ではサポートソフトとして主に Translation Manager (TM)を用いている。TM の場合 IU 全体のファイルは FXP のエクステンションを持ち、翻訳対は EXP のエクステンションを持つ。

⑤ TM 上での翻訳完了後、TM から元のフォーマットで IU を出力する。この出力した IU のフォーマットは開発側から渡った IU とまったく同じフォーマットである。英語のストリングは翻訳対象の言語たとえば日本語に置き換わっている。

⑥ 開発研究所では、この IU を受け取りその中に含まれているフォルダーまたはファイルをプログラムの適切な場所に格納する。ファイルの Encoding の処理は開発研究所もしくは TSC で行っている。開発側はさらにプログラムのビルド等を行う。

⑦ 最後に実際にプログラムを動かし翻訳のテストを行う。このフェーズを翻訳検証テスト(Translation Verification Test, TVT)と呼んでいる。このフェーズでは、開発研究所のテスト担当者と各国の TSC の TVT テスターが協力して翻訳の検証および修正作業を行う。TVT テスターが誤訳や不適切な翻訳を発見すると、④のところでファイルを修正し、再度⑤-⑦の作業を行い、修正を確認する。

TVT フェーズでは、④、⑤、⑥、⑦を繰り返し翻訳の修正を行っていく。本稿で述べる手法は通常の④から⑦の作業に⑧の作業を追加し、TVT の効率を上げる。

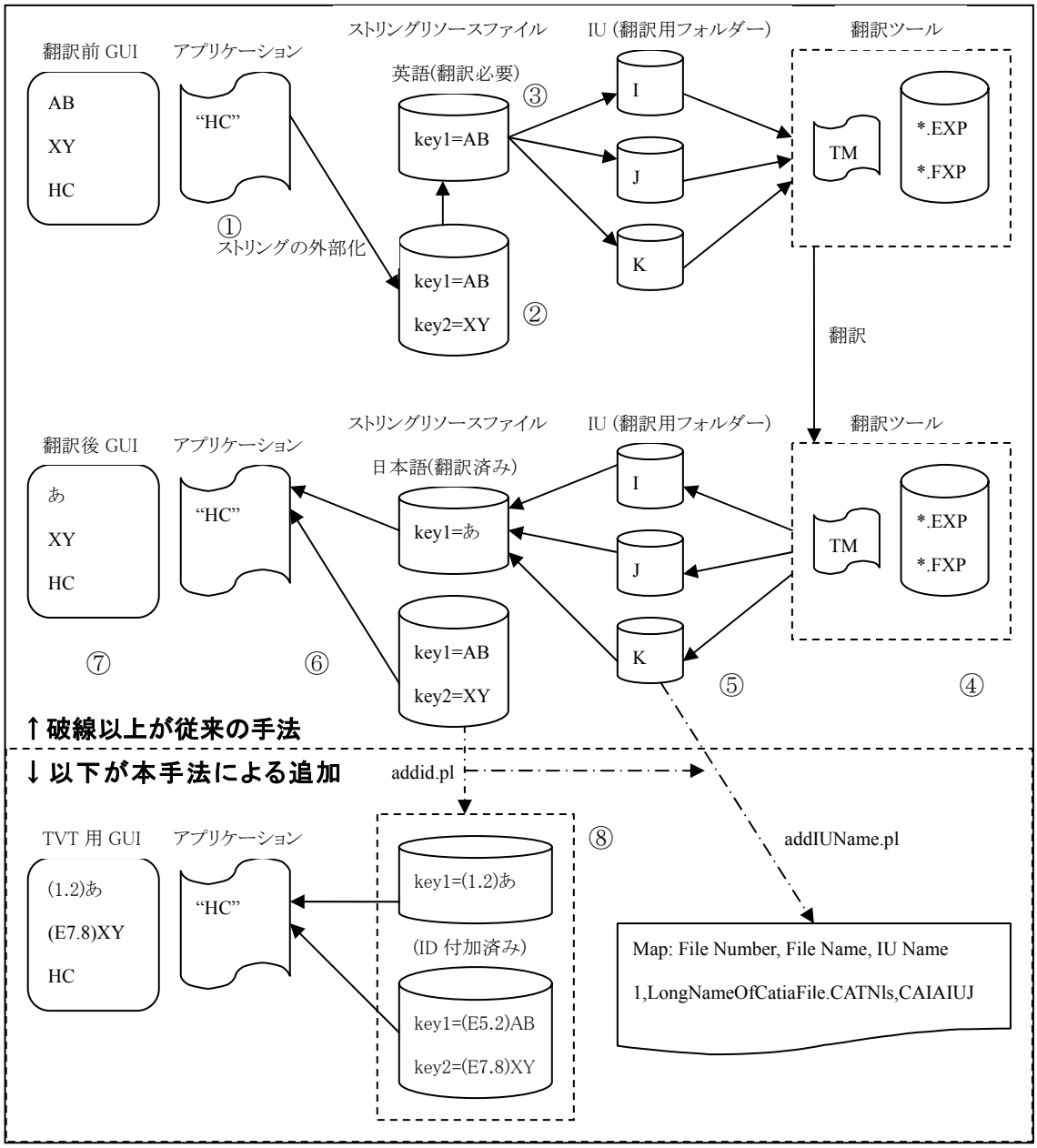

図 1 ストリングリソース翻訳の流れ

3. 問題点と研究の目的

3. 1 PII 翻訳時の問題

PII の翻訳は、通常の文書翻訳とは異なる PII 特有の問題を抱える。通常の文章は文脈の中で翻訳を行える。しかし、PII はプログラムから分離した外部テキストリソースファイル上で翻訳を行うので、文脈を理解した翻訳はできない。リソースファイル中のストリングがどのようなプログラムの文脈のなかで使用しているかは、外部化した翻訳リソースファイルからは知りえないからである。たとえば、"Update Warning"を翻訳するにも、「警告を更新」なのか「更新の警告」なのかは、プログラムの流れの中で確認せねばわからない。たとえ翻訳者にプログラムが渡されていても、どのような操作を行ったときこのストリングが出現するかはわからない。プログラムのソースファイルを参照しても、どちらが正しいかを決めるのは困難である。そこで各国の翻訳者は、開発研究所にそれぞれのストリングの意味を質問することになる。多くのやり取りの末、「警告を更新」なのか「更新を警告」なのかが判明する。翻訳に時間が十分ない時は、翻訳者は現在たぶんこちらだろうと思う方を選んで翻訳を完了し、TVT テスターに伝える。現実的には文脈のない短い何百、何千とあるストリングに対し、翻訳に確信がもてない場合をすべてリストアップすることは時間的に困難

なことも多い。それは、最終的には、リソースファイルにあるストリングの多くに注釈をつけることを求めるようなものであるからである。

3．2 TVT 時の問題

TVT テストは開発研究所が作成したシナリオに従って、プログラムの流れの中で、PII の翻訳が正しく行われているかを検証する。翻訳時とは反対に TVT テストにおいてはプログラムやそのプログラムの流れはわかる。しかし、そこに出力しているストリングがどのストリングリソースから出力しているかがわからない。翻訳済みのリソースファイルから出力しているのか、英語のリソースファイルから出力しているのか、リソースファイルからではなくプログラム中に埋め込まれた(hard code)ストリングから出力しているのかわからない。翻訳済みリソースファイルから出力していても、ストリングがどのリソースファイルのどのストリングから出力しているかがわからない。本研究はこの TVT テストにおける問題を解決することを目的とする。

TVT テスト中のこの問題はリソースファイルの規模が少しでも大きくなると、より大きな問題となる。たとえば、France Dassault 社の CAD/CAM プログラムである CATIA においては、リソースファイル数は約 8000 ファイルあり、キーのストリング数にすると 16 万 string 以上ある。そのうち 2004 年現在日本語に翻訳している PII ファイルは 6000 ファイルを超え、PII のリソースストリングは 10 万ストリングを超える。規模の問題とは別に、バージョンアップやリリースアップ時の翻訳では、新しい翻訳部分か古い翻訳部分かも見分けがつかない。

従来は TVT テスターが修正の必要な翻訳を発見したときは、GUI が表示したストリングをリソースファイル群の中から grep で探していた。2 つの異なる翻訳ストリングを連結して GUI に表示している時などは、いくら表示ストリングを探してもリソースファイル中のストリングにはマッチしないといったことも起こっていた。全く同一のストリングがリソースファイル中に複数存在する場合は、grep では GUI ストリングの場所を確定できない。その場合、個別にリソースストリングを書き換えて場所を特定する作業を行い、多大な時間を消費していた。従来 CATIA の場合 3 週間の TVT 作業中、日本語においては約4分の1の時間をストリング探索に使用していた。本研究はこの TVT テスターが抱える問題を解消することを目的とした。

次に、問題の解決のために導入したストリングリソース ID を CATIA への実装をもとに説明する。

4．翻訳検証手法の概要

図1を用いて、簡単に本手法を説明する。

1段目： 翻訳前にプログラムの GUI 上の出力には、「AB」「XY」「HC」の3つの英語のストリングがある。「HC」はプログラム中に埋め込まれたストリングで外部での翻訳は不可能である。「AB」「XY」はプログラム中に「key1」と「key2」が埋め込まれている。GUI 上で実際に表示するストリングは「key1=AB」「key2=XY」として外部のテキストファイルに分離している。
AB は翻訳対象ストリングなので、IU フォルダーの中に AB を含んだテキストファイルは入り、翻訳を実行する。

2段目： 翻訳したファイル「key1=あ」は日本語リソースファイルとしてシステムに組み込む。表示は「あ」「XY」「HC」となる。

3段目(破線以下の部分)： 2段目中央の翻訳を完了した日本語ファイルと外部化した英語ファイルに addid.pl プログラムでそれぞれ ID を振る(⑧)。このとき、addid.pl はファイル番号とファイル名のマップも同時に生成する。このマップにさらに addIUName.pl プログラムで処理しそれぞれのファイルに対応する IU フォルダー名を追加する。ディスプレイ上で ID 情報によりストリングの出所がわかる。「あ」は日本語1番ファイルの2行目、「XY」は英語7番ファイルの2行目、「HC」は hard code とわかる。

図1に現れる ID 追加前後のリソースファイル上でのストリングリソースを表1にまとめた。

表 1 ID 追加前と追加後のストリング

	ID 追加前	ID 追加後
日本語リソース	key1=あ	key1=(1.2)あ
英語リソース	key1=AB key2=XY	key1=(E5.2)AB key2=(E7.8)XY

5．実装でサポートしたリソースファイル

CATIA のリソースファイルは基本的に CATIA 特有の *.CATNls ファイルフォーマットで構成している。例外的に *.CATNls とファイル内のフォーマットは同一だがファイル名が*.txt になっているもの、および Java の properties ファイルも一部存在した。 *.CATNls ファイルは次のような平テキストフォーマットになっている。

key1=”Link Manager1”;

キーとキーのセパレータは「;」であり、ストリングは「”」で囲まれている。また、「” ”」内の改行を許している。キーとストリングのセパレータは=になっている。*.properties file 内では Java の properties ファイルのフォーマットを取る[4]。翻訳時には上記 Link Manager1 の部分が言語別のストリングに変わる。たとえば*.CATNls の key1 の例では、ファイル中で

key1=”リンク マネージャー1”;

となる。なお日本語ファイル中にキーがないと、通常プログラムは英語のキーのストリングを出力する。CATIA の英語のリソースファイルはインストールしたフォルダー中の

…¥resources¥msgcatalog
に存在し、日本語のリソースファイルは同じくフォルダーの
…¥resources¥msgcatalog¥Japanese
にある。CATIA の PII は上記フォルダー内のリソースファイルを切り替えプログラムを再起動するだけで、表示するリソースを切り替えることができる。

6. ストリングリソース ID フォーマット

実際にストリングリソース ID を CATIA のプログラムに付加した表示例を図 2 と図 3 に示す。図 2 では、「ｽﾀｰﾄ」の前には(5125.21)の ID を付加している。これは「ｽﾀｰﾄ」が 5125 番ファイルの 21 行目のキーのストリングから出力していることを示している。後述するプリフィックス(prefix)が無いので日本語リソースファイルからの出力であるとわかる。図3も同様であるが'E'のプリフィックスが付いているので、英語のリソースファイルからの出力であるとわかる。

図 2 CATIA ID 付き表示 （主に日本語ファイルから）

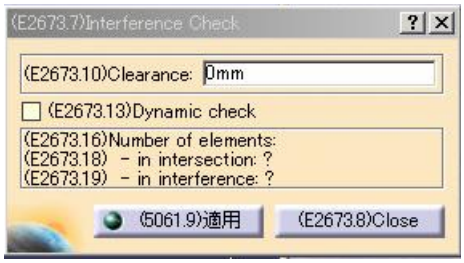

図 3 CATIA ID 付き表示 （主に英語ファイルから）

ストリングリソースIDは「( )」「Prefix」「Filenumber」「.」「Linenumber」の要素を持ち、次のフォーマットを取る。

(PrefixFilenumber.Linenumber)

このフォーマットは最初と最後のカッコ「( )」、プリフィックス(Prefix)、ファイル番号(Filenumber)、「.」ピリオド、行数(Linenumber)がそれぞれの役割をはたす。カッコは本来のメッセージストリングと ID を分離するために挿入している。またピリオドはキーのある行数(Linenumber)とファイル番号(Filenumber)を分離するために挿入している。プリフィックス(Prefix)は Filenumber 直前の任意の ASCII ストリングであり、addid.pl プログラム実行時に TVT テスターが決める。プリフィックスを TVT テスターが追加しなければ(Filenumber.Linenumber)となりプリフィックスは存在しない。なお、リソースファイルに ID を振るプログラムはファイル名と生成したファイル番号のマップテーブルを別途テキストファイルに生成する。TVT テスターはこのテーブルから目的のファイル名を知る。実際に前節の key1 に示したストリングに ID を付加した例を次に示す。

key1="(24.22)リンク マネージャー1";

このストリングを CATIA は GUI 上に

(24.22)リンク マネージャー1

と表示する。この例では、プリフィックス(Prefix)が存在せず、24 がファイル番号(Filenumber)、22 が行数(Linenumber)にあたる。このカッコ内の記述により TVT テスターは、これは日本語翻訳ファイルのファイル番号 24 番、22 行目にあるキーに対する翻訳であるとわかり、プログラムが別途生成したマップテーブルによりファイル名を特定し出力元のストリングを特定できる。必要があれば、TVT テスターはその特定した翻訳ストリングを修正する。

実装したプログラムでは、Filenumber と Linenumber を分離するピリオド(.)をキーの状況により他の ASCII 文字に変更する機能を入れた。たとえば、キーごとにアンダーバー(_)やコロン(:)に変更できる。これは旧バージョンと新バージョンのストリングリソースが存在する場合、ストリングをあらかじめ比較し、キーごとに分離する文字を指定するマップを作成する。この文字を指定するマップファイルがあるときは、マップテーブルが指定する文字をピリオドの変わりに使う。これにより TVT テスターは新しいキーと翻訳に変化があったものに注意を集中できる。

6. 1 プリフィックス(Prefix)

プリフィックス(Prefix)は出力元が原典の英語なら'E'、日本語なら'J'を振るかプリフィックスを振らないのを原則と

する。このプリフィックスにより TVT テスターは一目でストリングの出力元が英語のリソースファイルであるか日本語のリソースファイルであるかがわかる。

また、追加リソースファイル群がプログラム開発側から渡された時は"J1"等の付加によりストリングが追加リソース群からのものであることがわかる。別の用途としてはA翻訳者が翻訳したファイル群には'JA'、B 翻訳者が翻訳したファイル群には"JB"といったように TVT テスター側の意図を ID に盛り込むこともできる。リソースファイルの更新日時でファイルをソート可能なら翻訳時期により J の後のサッフィックスを変えれば、最近変更したファイルからのストリング出力であるといったこともわかる。 たとえば、図3ではEのプリフィックスにより、日本語のリソースファイル上のキーは存在しないけれども、英語のリソースファイル上には有ることがわかる。また図3の[適用]ボタンは英語のE2673とは別の翻訳ファイル 5061 の日本語済みストリングを出力している。日英にまたがったリソースファイルからの出力は不自然なので、TVT テスターは予測外のバグで日本語リソースファイルが壊れていないかなど 5061 のファイルとE2673 のファイルの内容を確認する必要があることがわかる。

6. 2 ファイル番号、行数

ファイル名ではなくファイル番号(Filenumber)を ID の構成要素にした理由は、GUI 上に表示する ID を含めたストリング長を短くするためである。実際のファイル名を使用すると表示ストリング(文字列)が長くなる。CATIA Version5 Release13 におけるファイル名の平均ストリング長は 28 文字で、キーの平均ストリング長は 34 文字である。ID 長 60 文字強となり、GUI の表示に破綻をきたす。本来のメッセージストリングがまったく表示されなかったり、ツールバーが何行にもなったり、ID 付の長いストリングにより本来表示している部分を隠したりする。ファイル番号も Hex 表示にするとさらに文字列は短くなる。しかし、テスターの利便性を考慮して通常の10進の番号にした。開発したID長は全体で高々10文字程度である。ID長はこのIDの最も重要な点で、ID を短くしなければ有効なツールとはなりえない。

キーも実際のキーのストリングを ID の構成要素にせずファイル中にキーが出現する行数(Linenumber)にした。理由は、ファイル名と同様にGUI上に表示するIDを短くするためと、TVTテスターが容易にテキストファイル中のキーの場所を発見できるようにするためである。

7. ストリングリソース ID 生成プログラム

本研究におけるリソース ID を追加するプログラム(addid.pl)は、Perl で実装した。処理可能なテキストフォーマットは *.CATNls と*.properties である。

addid.pl は次のオプションをとる。カッコ内はディフォルト値である。

-l　処理対象フォルダー名 (Japanese)
-f　処理対象ファイル名のフィルター (*)
-p　プリフィックス (Null)
-e　Encoding (System Default)

上記 Default 値を例とすると、addid.pl は次の実行結果を出力する。

(1) JapaneseWithID：
Japanese フォルダー内のファイルと同一のファイルでストリングに ID を追加したファイルを含むフォルダー
(2) JapaneseNumList.txt：
ファイル番号とファイル名のマップのリストファイル

引き続きJapaneseIUというフォルダーにIUのフォルダーをコピーしておき、addIUName.pl を Japanese の argument で実行すると、次の2つのリストを出力する。

(1) JapaneseNumListWithFolder.txt：
JapaneseNumList.txt のマップにIUのフォルダー名を追加したリストファイル
(2) JapaneseNumListNotInFolder.txt：
該当するIUフォルダーが存在しないファイル番号とファイル名のリストファイル

8. Java の properties file に対する適用

Java のプログラムに対しても同手法を適用し、ストリングに ID を追加表示することができる。

Java プログラムへの実装例を図4に示す。この図はデータ入力画面の一部で、プログラムはストリングを外部化するのに PropertyResourceBundle クラスを使用している。図中のストリングリソース ID の表示していない日本語ストリングはこのプログラムのデータである XML ファイルからの出力で TVT の範囲外の日本語である。「(1) Mon-Fri, Sat-Sun」には ID がなく hard code の出力例である。「(1)」は ID のパターンではないので本来のストリングとわかる。

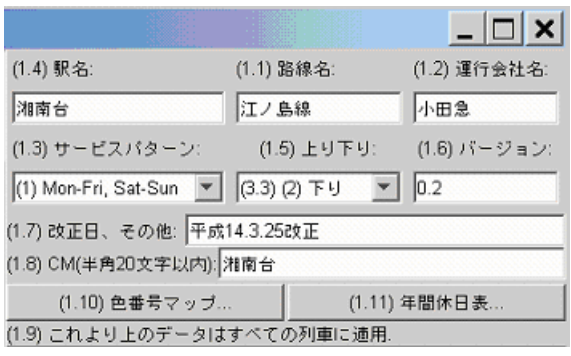

図 4: Java プログラムにストリングリソース ID を付加

Java においても CATIA と同様に本来の properties ファイルを ID 付ファイルに置き換えて Program 上に ID を表示することはできる。ところが、Java の場合はプログラム起動時に言語(language)、国(country)、および地域(variant)を指定できるので、その機能を活用すればファイルを置き換える必要はない。

Java はリソースファイル中の properties ファイル中のキーを探すとき、たとえば次の 1、2、3、4 の順で、キーを探索する。プログラムはキーが見つかったところでそのストリングを出力する。この例では、言語、国、地域をそれぞれ、日本語、日本、関西とした。

1. filename_ja_JP_kansai.properties
2. filename_ja_JP.properties
3. filename_ja.properties
4. filename.properties

そこで、仮想の国(00)、と仮想の方言(TVT)を考える。仮想の国ではキーに対応するストリングは日本語ではなく英語のストリングに ID を付加したストリングを持つ。同様に仮想の方言の地域では、日本語に ID を付加したストリングを表示する。なお(00)は(ゼロゼロ)である。
仮に abcd.properties(原典の英語)、abcd_ja.properties(翻訳した日本語)がストリング・リソース・ファイルとしてあるとすると、それらのファイルと追加するファイルは次のような関係にある。

abcd_ja_00.properties :
abcd.properties に対しストリング ID を付加したファイル

abcd_ja_00_TVT.properties :
abcd_ja.properties に対しストリング ID を付加したファイル

ID 付ファイルを追加するとリソースファイル全体は次のようになる。

(A) abcd.properties
(B) abcd_ja.properties
(C) abcd_ja_00.properties
(D) abcd_ja_00_TVT.properties

ファイル内の具体的なキーの内容は、たとえば次のようになる。

(A) key1 = Link Manager1
(B) key1 = リンクマネージャー1
(C) key1 = (E28.22) Link Manager1
(D) key1 = (24.22) リンクマネージャー1

通常の日本語の環境で Java を起動すれば(B)のリソースファイルを参照し日本語を表示する。(B)にキーがなければ(A)を参照する。Java の起動時に次のオプションを入れると、

-Duser.language=ja
-Duser.country=00
-Duser.variant=TVT

(D)の ID 付日本語ストリングリソースファイルを参照する。(D)にキーがなければ、(C)の ID 付英語ストリングリソースファイルを参照する。

上記のように Java の場合は CATIA のように、ファイルを置き換える必要はない。ストリングに ID の付いたファイルは上記のように Java の properties ファイルのファイル名のルールに従った TVT の仮想の国、TVT の仮想の地方の名前をつけて、本来のファイルと同じフォルダーに置くだけでよい。起動時のオプションを変更するだけで、通常のストリング出力と ID 付のストリング出力を変更できる。

9. ストリングリソース ID の特徴と効果

9.1 特徴

ストリングリソース ID には次のような特徴がある。

(1) この ID の最大の特徴は短いことである。短いがゆえに GUI 上に本来のメッセージに大きな影響を及ぼすことなく追加表示できる。
(2) ID を振るプログラム(addid.pl)は本体のプログラムとは独立に外部化したストリングリソースに対し ID を振っている。従って、テキストファイルにストリングリソースを外部化しているプログラムに対しては汎用的にこの手法を適用できる。
(3) また、この ID を振るプログラムはストリングリソースファイルのフォーマットのみを理解しておればよい。本体のプログラムに関しての知識は、リソースファイルのフォルダーの場所以外必要ない。
(4) 本体のプログラムに依存しないがゆえに TVT テスターは使い慣れた同一の手法でさまざまなプログラムに対して TVT を行うことができる。
(5) プリフィックスの導入により ID への TVT テスターの意思の反映も容易である。また「.」「( )」などに変化を持たせ、ID 長を長くせず TVT テスターの必要に応じて情報量を増加できる。
(6) 本手法およびプログラムは日本語以外の他言語にも活用できる。効果は翻訳言語数倍になる。

9.2 効果

具体的には、この TVT のための ID の導入により次のような効果がある。

(1) 従来は TVT テスターが grep を使って探していたストリングの出力元を一意に特定できるようになる。6000 ファイル以上を grep で search すると数十秒単位の時間が必要で作業効率を久しく低下していた。この ID の導入により出力元は一瞬でわかる。

(2) また、「key1=はい」、「key2=はい」といった複数の同一の翻訳が有るときにどちらのキーの側を修正すれば目的の表示を更新できるかと言ったことを知ることができる。従来は煩雑で時間のかかる作業を必要とした。

(3) プログラムが表示しているストリングを TVT テスターが hard code であるということを確定することは、上記(1)および(2)以上に容易ではなかった。ID の導入により、TVT テスターは ID を表示していないものは、日本語、英語を含めリソースファイル内のストリングではないと確定できる。通常そのストリングを外部ファイルに分離していない hard code ストリングと理解してよい。(図1 “Part1”, “Analysis1”参照、従来だとストリングリソースに Part1 と出てくる部分すべてを翻訳忘れでないか検討するので、TVT テスターの時間を浪費していた。)

(4) 複数ストリングリソースから組み合わせて1つの表示を作成している場合、従来は grep でリソースを検索し、出力元ストリングリソースを確定することは極めて困難であった。本 ID により、一つの GUI 上の表示に複数の ID を出力するのでストリングを連結した表示でもストリングリソースの出所を容易に知ることができる。

(5) TVT テスターがプリフィックスを決めたり、「.」セパレータの文字に特徴を持たせたり、TVT テスターの立場に立った情報を ID に含めうる。その結果より細かくストリングリソースの情報がわかり、従来発見が不可能であった問題を発見でき、TVT の作業効率ならびに翻訳の品質向上につながる。(図3の例)

(6) GUI 上に英語ストリングが出力しているときにも、一目で翻訳忘れなのか、翻訳対象外なのかわかる。(図3の例ではプリフィックス E により翻訳忘れでないことがわかる。)

(7) フォルダー名を含むファイル名を番号で表示したことにより GUI 内で同一のファイルからストリングを出力しているのか、異なるファイルから出力しているのか一目で理解できる。複数ファイルにまたがる場合、元のファイルの翻訳者が別人であったりする。同一人物でも統一性を顧慮しにくいので、このような場合 TVT テスターはより注意深く翻訳の検証を行う。

CATIA の TVT においては、本リソース ID の導入により、ストリングの探索に使用していた時間が TVT 全体の時間の約 1/4(約 30 時間)から 1/40 以下(1、2 時間)に減少した。これは日本語に限った効果なので、9カ国翻訳とすると効果は9倍にして勘案する必要がある。

最後に、本手法は PII の翻訳時の問題を解決できる可能性がある。プログラムの操作やテストシナリオに従ってプログラムをストリング ID 付き英語で操作する。このとき出力する英語をプログラムの流れを理解しながら翻訳する。翻訳者は出力している ID からリソースファイルの該当する部分を翻訳する。プログラムの流れを理解しながら翻訳できるので PII 翻訳の品質は改善する。ただし、この方法ではファイルごとに翻訳は完了しないけれども、シナリオがある部分に対しては翻訳品質の向上が期待できる。

10. 結論

PII の翻訳は2つの大きな問題を抱える。一つは翻訳時に文脈(プログラム操作の流れ)がわからないことと、TVT(Translation Verification Test)時ストリングの出力元がわからないことである。本研究では後者の問題の解決を行った。問題解決のため、本体のプログラムからは独立に外部ストリングリソースファイル中の全ストリングリソースの前に、TVT テスターに有用でコンパクトな ID を振った。これによりプログラムの実装に関する知識をもたない TVT テスターが、GUI 上のストリングの出力元となるストリングリソースの場所(ファイル名とキーのある行数)を容易かつ確実に特定できるようになった。また同時に従来は TVT テスターには確定困難であった外部化していない hard code ストリングをも TVT テスターが確定できるようになった。

謝辞

本研究では、TVT テスターの濱畑健次氏の多大なる協力を得た、ここに謹んで感謝の意を表する。

参考文献

[1]Testing your internationalized Eclipse plug-in http://www-106.ibm.com/developerworks/opensource/library/os-i18n2/, July 2002

[2]Java Tutorial: Internationalization, http://java.sun.com/docs/books/tutorial/i18n/intro/index.html, June 2004

[3]Nadine Kano, Developing International Software for Windows95 and Windows NT, Microsoft Press, 1995.

[4]Andrew Deitsch and David Czarnecki, Java Internationalization, O’REILLY, 2001

[i] CATIA®は Dassualt Systems の登録商標です。