(19)日本国特許庁(JP)　(12)特　許　公　報(B2)　(11)特許番号
特許第6907305号
(P6907305)
(45)発行日　令和3年7月21日(2021.7.21)　(24)登録日　令和3年7月2日(2021.7.2)

(51)Int.Cl.　F I
G06F 8/76 (2018.01)　G06F 8/76

請求項の数 5　(全 14 頁)

(21)出願番号　特願2019-515408(P2019-515408)
(86)(22)出願日　平成29年9月20日(2017.9.20)
(65)公表番号　特表2019-530927(P2019-530927A)
(43)公表日　令和1年10月24日(2019.10.24)
(86)国際出願番号　PCT/US2017/052487
(87)国際公開番号　WO2018/057605
(87)国際公開日　平成30年3月29日(2018.3.29)
審査請求日　令和2年1月14日(2020.1.14)
(31)優先権主張番号　62/397,051
(32)優先日　平成28年9月20日(2016.9.20)
(33)優先権主張国・地域又は機関
米国(US)

(73)特許権者　518279875
イングラム　マイクロ　インコーポレーテッド
アメリカ合衆国　９２６１２、カリフォルニア州、アーバイン、スイート　１００、ミッチェルソン　ドライブ　３３５１
(74)代理人　100138760
弁理士　森　智香子
(72)発明者　パヴェル　ブグロフ
ロシア国　１１５４７０、モスクワ、ナガチンスカヤ　ナベレジナハ　４６、アパート　１１９

審査官　石川　亮

最終頁に続く

(54)【発明の名称】地域化されていない行の自動検出、修正、及び翻訳方法

(57)【特許請求の範囲】
【請求項１】
地域化されていない行の自動検出、修正、及び翻訳方法であって、
（ａ）少なくとも１つのソースファイルを受信するステップと、
（ｂ）前記少なくとも１つのソースファイルから少なくとも１つのメッセージファイルを少なくとも１つのローカリティについて生成するステップであって、前記少なくとも１つのメッセージファイルは、少なくとも１つのメッセージ識別情報、前記少なくとも１つのソースファイルにおける前記メッセージ識別情報の少なくとも１つの位置、少なくとも１つの翻訳されていない文字列と、少なくとも１つの翻訳されていない文字列に対応する少なくとも１つの翻訳を含む、ステップと
（ｃ）前記少なくとも１つのメッセージファイルをそれぞれ構文解析し、前記少なくとも１つのメッセージ識別情報を取得して、前記少なくとも１つのメッセージ識別情報がメッセージ識別情報データベースに既に存在するかを確認するステップと、
（ｄ）前記少なくとも１つのソースファイルにおける前記少なくとも１つのメッセージ識別情報のコンテキストを少なくとも用いて、複数の地域化パターンを生成して前記メッセージ識別情報データベースに保存するステップと、
（ｅ）前記少なくとも１つのメッセージ識別情報を確認して、少なくとも１つの他のローカリティに存在するかを確認し、そうであれば、ステップ（ｇ）に進み、そうでなければ、ステップ（ｆ）に進むステップと、
（ｆ）前記少なくとも１つのメッセージ識別情報に基づいて追加のローカリティを延伸

させ、次の少なくとも１つのメッセージ識別情報を確認するためにステップ（ｅ）に戻るステップと、
（ｇ）翻訳エンジンに翻訳を要求し、前記翻訳エンジンから翻訳されたコード文字列を受信し、前記メッセージ識別情報データベースに前記翻訳されたコード文字列を記録するステップと、
（ｈ）レビュー関係者へのレポートを作成するステップと、
（ｉ）ＧＵＩを介したプレゼンテーションのための複数のプレゼンテーション用ファイルを生成するステップと、
（ｊ）前記複数のプレゼンテーション用ファイルが前記少なくとも１つのローカリティについて同一のメッセージ識別情報及び翻訳されたコード文字列を有するかを検証するために、前記複数のプレゼンテーション用ファイルを構文解析するステップと、
（ｋ）前記メッセージ識別情報及び翻訳されたコード文字列が利用不可である場合に前記複数のプレゼンテーション用ファイルを更新するステップと、
（ｌ）間違いを修正し、前記レビュー関係者に変更について報告するステップと、
を含む、方法。
【請求項２】
全ての存在するローカリティについてＪＳＯＮファイルを構文解析し、レポートを送信して該レポートに基づいてエラーを処理することにより、ＪａｖａＳｃｒｉｐｔ　Ｏｂｊｅｃｔ　Ｎｏｔａｔｉｏｎ（ＪＳＯＮ）に対して前記メッセージファイルをバリデートすることを更に含む、請求項１に記載の方法。
【請求項３】
地域化パターンを取得して、それらを前記データベースに保存することを更に含む、請求項１に記載の方法。
【請求項４】
前記少なくとも１つのメッセージファイルはｅｎ＿ＵＳ．ｐｏファイルである、請求項１に記載の方法。
【請求項５】
前記翻訳は、翻訳ユーティリティアプリケーションプログラミングインターフェースを用いて要求される、請求項１に記載の方法。
【発明の詳細な説明】
【技術分野】
【０００１】
（関連出願の相互参照）
本出願は、２０１６年９月２０日に出願された米国特許出願第６２／３９７，０５１号の国際出願であり、その優先権の利益を主張するものであり、その本文及び図面はその全体に亘って本明細書に参照により援用される。
【０００２】
本発明は、地域化されていないコード行の自動検出方法に関し、より具体的には、アプリケーションパッケージングスタンダード（Application Packaging Standard）（ＡＰＳ）対応のアプリケーションインターフェースにおけるコード行の検出、修正及び翻訳に関する。
【背景技術】
【０００３】
サービス自動化システムは、アプリケーションパッケージングスタンダード（ＡＰＳ）に対応する多数のアプリケーションを必要とする。アプリケーションパッケージングスタンダード（ＡＰＳ）とは、アプリケーションソフトウェアをホスティングプラットフォームと一体化させるための技術を定義する規格である。かかるホスティングプラットフォームをあるアプリケーションと一体化させることは、そのアプリケーション用にＡＰＳパッケージを生成することによって実行され、この場合、そのアプリケーションはＡＰＳアプリケーションと呼ばれる。ＡＰＳプラットフォーム上で開発するで、アプリケーションを

異なるホスティングプラットフォームで広く使うことが可能になる。かかるアプリケーションはまた、母国語及び／又は方言で、世界中の様々な地域でも展開されている。かかるアプリケーションが使用されるためには、時にそのグラフィカルユーザインターフェース（ＧＵＩ）が現地の言語を用いて展開される。例えば、米国（Ｕ．Ｓ．）にいるユーザに対してはＧＵＩが英語で提示されるテキスト要素を有し得るのに対し、同じＧＵＩ要素がロシアにいるユーザに提示されるときには同じテキストをロシア語に翻訳してもよい。ＧＵＩ要素に（英語のみに対して）現地の言語を用いることで、カスタマイズされ、かつ改善されたユーザ体験を可能とする。

【０００４】

国際化（ｉ１８ｎ）は、文字列及び他のロケール特有部分（日付又は通貨形式）をアプリケーションから抽出することができ、容易に言語及び文化に地域化できるアプリケーションを開発するプロセスである。地域化（ｌ１０ｎ）は、抽出された部分について翻訳及び地域化された形式を提供することによって、それらの利用可能性を特定の文化又は言語市場で有効とするために、アプリケーション及びテキストを適合させるプロセスである。例えば、米国のアプリケーションを豪州又は英国のユーザにアクセス可能にするには、いくつかのスペル修正程度の修正を必要とするかもしれない。しかし、米国のアプリケーションを日本人のユーザに使用可能にする、又は韓国のアプリケーションをドイツ人のユーザに使用可能にするには、ソフトウェアが異なる言語で動作することだけではなく、異なる入力技術及び提示規定を使用できるようにする必要がある。

【０００５】

各言語及び各アプリケーション用にＧＵＩ（及びその中のＧＵＩ要素）を開発することは、実行可能な提案ではない。いわゆる「言語ネイティブ」のＧＵＩを開発するために必要な時間、労力及び費用は、世界中の複数の地域でアプリケーションを展開しようとするときには負担となり得る。

【０００６】

この負担を減らすために、ソフトウェア及び開発プラットフォーム、例えば、ＡＰＳプラットフォームにより、ＧＵＩ要素を地域化して翻訳することが可能になる。これによって、ＧＵＩは、レンダリングされたときに、その地域（又はユーザ）にネイティブである言語であることが可能になる。しかしながら、ＧＵＩインターフェースの従来の翻訳及び変換は一貫して提示されていない。これは、一つの言語のアルファベットを構成する文字のセットが別の言語と異なるためである可能性がある。空間特性の変化は、翻訳言語におけるテキストをより大きく（又はより小さく）させる、あるいは意味が「翻訳の中で失われる」。

【０００７】

更に、ＧＵＩが開発又は改善されると、エンドユーザに対して画面でレンダリングされるデータ量は、ユーザがＧＵＩの一部として見る任意のテキストメッセージを含み、増加し得る。特に、行の数、インターフェース要素のマーク、及び画面様式は増加する傾向にある。ＡＰＳ規格によれば、全てのアプリケーションインターフェース要素は地域化及び国際化されなければならない。しかしながら、インターフェース行及び要素は、アプリケーションコードに含まれた後に必ずしも適切にフォーマットされているわけではない。結果的に、エンドユーザはアプリケーション内で、部分的な地域化及びメッセージの翻訳によって引き起こされる欠陥を観察することができる。

【０００８】

例えば、図１を参照すると、概して１００にて、ＡＰＳ対応アプリケーションの地域化されていない行の翻訳の従来の方法の実施形態が示されている。従来の方法は、ソースコードを収集するステップ１０２と、ｍｓｇｍａｋｅポストを通して地域化をインポートするステップ１０４と、ポータブルオブジェクト（．ｐｏ）メッセージファイルを生成するステップ１０６と、ｍｓｇｍａｋｅポストを、地域化されていない行の翻訳をドラフトするテクニカルライターに転送するステップ１０８と、翻訳を追加するステップ１１０と、地域化されたメッセージをＪａｖａＳｃｒｉｐｔ　Ｏｂｊｅｃｔ　Ｎｏｔａｔｉｏｎ（Ｊ

ＳＯＮ）にてエクスポートすることでＡＰＳパッケージを構築するステップ１１２とを含み、結果的に国際化ファイルを有するＡＰＳパッケージを有する１１４に至る。

【０００９】

従来の方法１００では、全ての行が、提示されて正しくインターフェースに地域化できるわけではない。その理由としては、これらに限定されないが、ローカリティの間違ったフォーマット、プロセッサエラー、間違ったキー、利用可能な翻訳の不足、又は追加ローカリティに関する誤記若しくは欠如ファイルを、いくつかの非限定的な例として挙げることができる。これは結果的に、必要なシナリオの不完全な実施、並びに確認及び検証に伴う高費用を引き起こす。

【００１０】

理想的には、上記の問題をアプリケーション開発段階で防止することができる。更に、ＧＵＩ要素の翻訳の正確さの自動及び半自動での確認が所望される。これによって、確認の改善および、手動での検証に関連するコストの軽減の助けとなるであろう。最後に、アプリケーションライフサイクル管理内での地域化エラーのタイムリーな検出方法が所望される。

【００１１】

したがって、地域化されていない行の自動検出、修正、及び翻訳をするための方法の需要がある。

【発明の概要】

【００１２】

このことを受けて、本開示は（以下に更に記載するとおり）従来技術の1つ以上の不利益を実質的に防止する、地域化されていないコード行の自動検出、修正及び翻訳の方法を対象としている。本開示の一様態では、組み立てられたアプリケーションがメッセージファイル（＊．ｐｏ）に組み込まれる前に、アプリケーションソースコードにおける地域化された行を周期的に確認する方法が提供される。別の実施形態では、全てのサポートされるアプリケーション言語について、行の自動翻訳が提供される。ｉ１８ｎ　ＪＳＯＮファイルで提供される行及びそれらの翻訳は、メッセージファイルに対してバリデートされる。例示的な実施形態によれば、地域化されていない行は、地域化マーカーを用いてソースコード内で確認される。ＡＰＳによれば、ユーザインターフェースを組み立てるためのａｐｓコマンド行ツールのセットから地域化データを導き出すために、ａｐｓ　ｍｓｇｍａｋｅユーティリティについて特別なトラップ（すなわち、ホック）が使用される。

【図面の簡単な説明】

【００１３】

【図１】テクニカルライターによって手動でコード行が翻訳される既存プロセスの概略図を示す。

【図２】地域化されていない行の自動検出、修正、及び翻訳のための提案される方法の概略図を示す。

【図３】地域化パターンの自動検出及び収集のための方法のフローチャートを示す。

【図４】地域化されていない行の自動検出、修正、及び翻訳のための方法のフローチャートを示す。

【図５】自動検出、及びプロパゲーション行のバリデーションのための方法のフローチャートを示す。

【図６】地域化されていない行の自動検出、修正、及び翻訳のための方法の概略図を示す。

【発明を実施するための形態】

【００１４】

これより本発明の好適な実施形態を詳細に参照し、それらの実施例は添付図面に図示されている。

【００１５】

本開示の少なくとも１つの実施形態では、組み立てられたアプリケーションがメッセージファイル（＊．ｐｏ）ファイルに組み込まれる前に、アプリケーションソースコード内で地域化された行を周期的に確認する方法が提供される。別の実施形態では、全てのサポートされるアプリケーション言語についての、行の自動国際化が提供される。行及びそれらの翻訳は、メッセージファイル（＊．ｐｏ）に対してバリデートされている。

【００１６】

例示的な実施形態によれば、図２を参照すると、地域化された行は地域化マーカーを用いてソースコード２０２内にマーキングされている。ＡＰＳを用いる実施形態では、ユーザインターフェースを組み立てるためのａｐｓコマンド行ルールのセットから地域化データを導き出すために、ａｐｓ　ｍｓｇｍａｋｅユーティリティ２０４について特別なトラップ（すなわち、ホック）が用いられる。ａｐｓ　ｍｓｇｍａｋｅコマンドは、言語ごとに１ファイル、翻訳文字列をメッセージファイル（＊．ｐｏ）に抽出する。．ｐｏメッセージはプレーンテキストファイルであり、所定の言語での全ての利用可能な翻訳文字列及びそれらの解釈（翻訳）を含む単一の言語を表す。このファイルは、翻訳者が対象言語にて翻訳文字列の解釈を提供するための便利な方法である。．ｐｏメッセージファイルは多くのエントリからなり、各エントリは元の翻訳されていない文字列とその対応する翻訳との関係性を説明する。

msgid “[元の翻訳文字列がこちら]”
msgstr “[翻訳された文字列がこちら]”

【００１７】

上記例を用いて、スペイン語での翻訳を以下のとおりに表すことができる。

```
msgid "Diskspace - Usage Only"
msgstr "Espacio en disco - Solamente Uso"
```

【００１８】

例示的な実施形態によれば、ホック（すなわち、地域化マーカー）の２つの例示的なフォーマットが用いられる。

【表１】

ＪａｖａＳｃｒｉｐｔファイル（＊．ｊｓ）	Ｊａｖａファイル（＊．ｊａｖａ）
_(“メッセージ”)	__(“メッセージ”)

【００１９】

＊．ｊａｖａのアンダーバーの対及び＊．ｊｓの単一のアンダーバーの機能は、翻訳用に文字列を利用可能にするためのホックである。例えば、__(“翻訳する文字列”)は、ａｐｓ　ｍｓｇｍａｋｅコマンドに、．ｐｏファイル２０６内にmsgid “翻訳する文字列”及びmsgstr “翻訳された文字列”の対を生成させる。一般的な場合では、文字列には翻訳されるべきでないマッピングパラメータを含んでもよい。マッピングされたパラメータは、一対のアンダーバー“__”の間に包含されなければならない。以下の例では、ｍｓｇＫｅｙ文字列がパラメータを含み、これらの値はｐａｒａｍＯｂｊｅｃｔｓマッピング文字列で見出されなければならない。

```
_(msgKey, paramObjects)
```

【００２０】

地域化マーカーは、ショート又はフルフォーマットで使用できることを更に理解されたい。例えば、

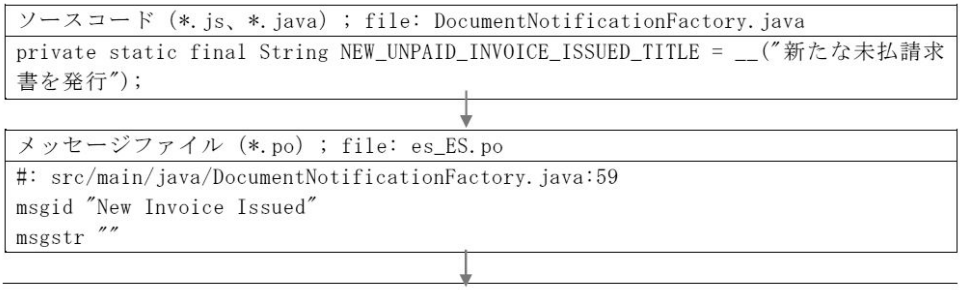
ＪａｖａＳｃｒｉｐｔ（＊．ｊｓ）：
ショートフォーマット：_("User __username__ created")
フルフォーマット：_("User __username__ created", {"username":"John Smith"})
Ｊａｖａ（＊．ｊａｖａ）：
ショートフォーマット：__("User __username__ created")
フルフォーマット：__("User __username__ created", {"username":"John Smith"})

【００２１】

よって、これらの行の全てが．ｐｏメッセージファイルに追加され、続いて以下のとお

りＪＳＯＮにも追加される。

【表２】



【００２２】

__()関数は、文字列@ｍｓｇＫｅｙを地域化するために使用される。例えば、@ｍｓｇＫｅｙは実際のところ、文字列" __itemsCount__個のアイテムを発見"である。この文字列はｉｔｅｍｓＣｏｕｎｔパラメータを含み、一対のアンダーバーで囲まれる。パラメータは、マッピング文字列 {"itemsCount":counter} によって定義される。そのため、ａｐｓ　ｍｓｇｍａｋｅがＪａｖａＳｃｒｉｐｔ（登録商標）コードを構文解析するときに以下のコードを満たす場合：

_("__itemsCount__個のアイテムを発見", {"itemsCount":itemsCount})

これは．ｐｏメッセージファイルに以下の一対の記録を自動的に生成する：

msgid "__itemsCount__個のアイテムを発見"

msgstr ""

【００２３】

結果として得られる．ｐｏメッセージファイル２０６は正しいと考えられ、地域化サンプル（すなわち、地域化パターン）の辞書のための基礎として用いられる。コードの可変性が割と高いため、商品ソースコードで地域化される行を高い正確性で判定することは不可能である。例示的な実施形態によれば、ｅｎ＿ＵＳ．ｐｏのネイティブローカリティの．ｐｏメッセージファイルに書き込まれているキーのソースコードの行は、地域化サンプルとして自動的に考慮及び収集される。

【００２４】

図２に示すように、コード２０２の様々なソースをレビューするのにソースコードアナライザ２０８を用いてもよい。地域化マーカーを有する行が見つかると、ソースコードアナライザ２０８は現在の行内で地域化マーカーに先行する全てをデータベース（すなわち、データベース６５６）に保存する。

【表３】

パターン	タイムスタンプ	ファイル拡張子
label:	2016-04-10 0:00:00.000	JS

【００２５】

上記記録は地域化パターンと呼ばれる。例えば、以下：

var warn =_("使用を開始するにはserviceNameサービスを自分自身に割り当てること。", {serviceName: serviceConstants.SERVICE_NAME}では、地域化サンプルは"var warn = "となる。

【００２６】

例示的な実施形態によれば、地域化サンプル（パターン）は複数の規則に基づいて生成される。本開示の少なくとも１つの実施形態では、＊．ｊａｖａファイルにおける行がテキスト定数の宣言（すなわち、テキスト“static final String”又は“final static”）を含む場合、アプリケーションは“String（文字列）”の実際の言葉を含む文字列の前に位置する全てを地域化パターンとして考える。よって、文字列public final static String PASSWORD_NOT_CHANGED = __("パスワードは変更されていません")については、地域化パターンはpublic final static Stringである。

【００２７】

地域化マーカーの前にテキストがない場合（例えば、複数行の文字列において）、ソースコードアナライザ２０８は１つ上の行にある全てを地域化パターンとして捉える。上の行が地域化マーカーを有する文字列を含む場合、アプリケーションは次に上にある行に移動する。例えば、以下の行では：

【表４】

```
return runNotification(apsAccountUuid, apsUserUuid, () -> {
return buildNotification(apsAccountUuid, apsUserUuid, entity, domainName,
NotificationMessageStatus.inProgress,
__("ドメイン割り当て"),
__("ドメイン割り当て \"{domain}\" は進行中。"),
__("ドメイン割り当て \"{domain}\" は進行中。" +
"完了するのに数分かかる場合があります。"))
.send();
});
```

地域化パターンは以下となる：

return buildNotification (apsAccountUuid, apsUserUuid, entity, domainName, NotificationMessageStatus.inProgress)。

【００２８】

既に生成された地域化パターンに基づいた検索中に、複数行の文字列に出くわした場合、ソースコードアナライザ２０８は、文字列の終わりと解釈されるセミコロンまで文字列を読み取る。他の境界線全ての場合はコードレビュー中に解決されるため、地域化パターン生成のメカニズムは同一規格内で実行される。一例示的な実施形態によれば、地域化マーカーは、地域化パターンのリストが形成された後に自動的に追加される。地域化パターンのリストが形成された後、ソースコードアナライザ２０８は、保存された地域化パターンを用いて、特定のファイル拡張子（例えば、ＪＳ）を有する全てのファイルにおいてエントリを探し求める。例えば、サンプルlabel：マーカーがある場合、サンプル文字列マーカー（例えば、二重引用符）がある行の終わりを探し、続いて地域化マーカーが存在しなかった場合にはこの文字列を地域化マーカーで延伸させる。例えば、

Label: "テスト文字列" ---> label:_("テスト文字列")

【００２９】

続いて、 図２に示すように、ａｐｓ ｍｓｇｍａｋｅツールを介して、行プロパゲーションプロセス（地域化マーカーを有する全てのコード行がＰＯファイルに移動される）が実行される。ステップ２０４では、ａｐｓ ｍｓｇｍａｋｅはメッセージファイル（＊．ｐｏ）内の引用符内に配置されたコンテンツを配置するのを促進し、それが地域化キーとなる。

【００３０】

全ての地域化キーがネイティブローカリティの．ｐｏメッセージファイル内（例えば、"ｅｎ＿ＵＳ．ｐｏ"）に追加された後に、ロケール解析が開始される（２１０）。ロケールアナライザは、ネイティブ言語＊ｐｏメッセージファイルを、全ての存在するｍｓｇＩＤについて構文解析する。その後、該ツールはネイティブと追加言語ファイルとを比較し、内部に存在していないｍｓｇＩＤで追加言語ファイルを延伸させる。次に、行の国際化プロセス２１２が、ｇｅｔｔｅｘｔファイル（＊．ｐｏ）２１４と作用するように構成される利用可能なライブラリを利用して開始される。なお、ｇｅｔｔｅｘｔファイルはマルチリンガルなプログラムを書くために一般的に使用される国際化及び地域化（ｉ１８ｎ）システムに基づくことを理解されたい。例えば、プログラミング言語Ｐｙｔｈｏｎについては、「Ｐｏｌｉｂ」ライブラリを使用することができる。 行の自動翻訳については、Ｇｏｏｇｌｅ Ｔｒａｎｓｌａｔｅ ＡＰＩを使用することができる。

【００３１】

各ｍｓｇＩＤ（＊ｐｏファイル内の文字列はキーからなる）については、翻訳要求は、１つの非限定的な例を挙げると、Ｇｏｏｇｌｅ Ｔｒａｎｓｌａｔｅ ＡＰＩを使用する

Ｇｏｏｇｌｅ　Ｔｒａｎｓｌａｔｅなどの翻訳エンジンに、ＪＳＯＮフォーマットで送信される。

【００３２】

要求の実行が成功した場合には、サーバは「ＯＫ」の応答と、ＪＳＯＮでの翻訳結果を返す：

```
200 OK
{
  "データ": {
    "翻訳"：[
     {
           "TranslatedText": "Nueva factura emitida"    （スペイン語）
        }
    ]
  }
}
```

【００３３】

例示的な実施形態によれば、翻訳はＡＰＳアプリケーションによってサポートされる全ての言語について要求され、又はシステム内に．ｐｏメッセージファイルを有する言語のみについて要求される。翻訳の選択は、国際化文字列起動モードに基づいて行われる。翻訳された行は、以下のフォーマットでデータベース内（すなわち、データベース６５６）にも書き込まれる。

【表５】

キー	値	タイムスタンプ	ロケールコード
New Invoice Issued	Nueva factura emitida	2016-04-10 0:00:00.000	es_ES
New Invoice Issued	Nieuwe factuur Uitgegeven	2016-04-10 0:00:00.001	de_DE

【００３４】

例示的な実施形態によれば、適切な翻訳された行は適切なｇｅｔｔｅｘｔファイルに書き込まれる。次に、翻訳バリデーションマニュアルプロセス２１８が開始される。担当テクニカルライターは、自動翻訳された文字列をレビューする。

【００３５】

ステップ２２０では、ＡＰＳアプリケーション（すなわち、ＡＰＳパッケージ２２２）が組み立てられた後、メッセージファイル（＊．ｐｏ）のコンテンツは地域化ファイル＊．ｊｓｏｎに変換される。

【００３６】

例示的な実施形態によれば、行は．ｐｏメッセージファイルとＪＳＯＮ２２４との間でバリデートされる。パッケージが組み立てられた後、バリデーション文字列は、全ての地域化キーと、対応する地域化値とを、．ｐｏメッセージファイルからインポートする。次に、文字列は同一名及びファイル拡張子＊．ｊｓｏｎを有するファイルを確認する。続いて、文字列は、メモリにロードされたキーと値とを、ＪＳＯＮファイル内のキーと値とで比較する。キー又は値が見つからなかった場合は、自動でのエラーレポートが生成され（２２６）、１つの非限定的な例を挙げるならば、アプリケーション開発者に送信されてもよい。

【００３７】

ここで図３を参照すると、本開示の一実施形態による、データベースを地域化サンプルで事前設定するソースコードにおける地域化されていない行を確認するための方法のフローチャートが示されている。ステップ３０５では、．ｐｏメッセージファイルは、上に開示するとおり、ａｐｓ　ｍｓｇｍａｋｅツールによって生成されている。本開示の少なくとも１つの実施形態では、ａｐｓ　ｍｓｇｍａｋｅコマンドは、一言語につきファイル１

つずつ、翻訳文字列を．ｐｏメッセージファイルに抽出する。このファイルは、翻訳者が翻訳文字列の解釈をターゲット言語に提供するための便利な方法である。ａｐｓ　ｍｓｇｍａｋｅユーティリティは、ＧＵＩを組み立てるためのａｐｓコマンド行ツールのセットから地域化データを導き出すために、特別なトラップ（すなわち、ホック）を使用する。

【００３８】

ステップ３１０では、ネイティブローカリティの．ｐｏメッセージファイルが構文解析される。例えば、ｅｎ＿ＵＳ．ｐｏは、米国（ＵＳ）における英語言語（ｅｎ）についてのローカリティ．ｐｏメッセージである。この．ｐｏメッセージファイル名はｌｌ＿ＣＣ．ｐｏの形態にある。２文字のプライマリコード（ｌｌ）は、ＩＳＯ６３９－１の言語仕様書によって定義されている。２文字サブコード（ＣＣ）は、ＩＳＯ３１６６－１の国仕様書に基づいて解釈される。言語部分は必ず小文字で書き込まれ、国部分は大文字で書き込まれる。セパレータはアンダーバー（「＿」）である。本開示の少なくとも１つの実施形態では、．ｐｏメッセージファイルはプレーンテキストファイルであり、所定の言語にて全ての入手可能な翻訳文字列及びそれらの解釈（翻訳）を含む単一の言語を表す。なお、．ｐｏメッセージファイルは、多くのエントリからなり得て、各エントリは元の翻訳されていない文字列とその対応する翻訳との間の関係性を説明することを認識されたい：

msgid “元の翻訳文字列がこちら”

msgstr “翻訳語文字列がこちら”

【００３９】

ステップ３２０では、プロセスは文字列ｍｓｇｉｄが既にデータベース（例えば、データベース６５６）に存在するかを判定する。ｍｓｇｉｄが存在する場合、プロセスはステップ３１０に戻る。そうでなければ、プロセスはステップ３３０で地域化パターンを集める。次に、プロセスは地域化パターンをデータベース６５６に保存し、ステップ３４０で、例えばテクニカルライターなどの責任関係者にレポート２１２を送信する。なお、レポート２１２は新たに加えられた地域化パターンを含み、これは開発者又はテクニカルライターによって除外し、かかる文字列（複数可）を地域化パターンとして認識しないことを例外とするように、解析可能であることを認識されたい。ステップ３５０では、プロセスは．ｐｏメッセージファイルのファイルの終端（end of file）（ＥＯＦ）に到達したかを判定する。ＥＯＦに到達している場合、プロセスはステップ３６０で終了する。そうでなければ、プロセスはステップ３１０に戻る。

【００４０】

ここで図４を参照すると、本開示の一実施形態による、データベースを地域化サンプルで事前設定するソースコードにおける行の自動国際化のための方法のフローチャートが示されている。ステップ４０５では、．ｐｏメッセージファイルは、上記するように、ａｐｓ　ｍｓｇｍａｋｅツールによって生成される。少なくとも１つの実施形態では、＊ｐｏファイルは、ソースコードアナライザの作動後（ステップ２０４の２度目の繰り返し）の前の段階から取得することができる。本開示の少なくとも１つの実施形態では、ａｐｓ　ｍｓｇｍａｋｅコマンドは翻訳文字列を、各言語につき１ファイル、．ｐｏメッセージファイルに抽出する。このファイルは、翻訳者がターゲット言語で翻訳文字列の解釈を提供するための便利な方法である。ａｐｓ　ｍｓｇｍａｋｅユーティリティは、ＧＵＩを組み立てるためのａｐｓコマンド行ツールのセットから地域化データを導き出すために、特別なトラップ（すなわち、ホック）を使用する。

【００４１】

本開示の少なくとも１つの実施形態では、プロセスは自動的に、翻訳が望まれる各ローカリティについて、複数の．ｐｏメッセージファイルを生成する。なお、各ローカリティについての．ｐｏメッセージファイルは、翻訳する必要がある各ｍｓｇｉｄを含むことを更に認識されたい。ステップ４１０では、ステップ３１０に記載されるとおり、ネイティブローカリティの．ｐｏメッセージファイルが構文解析される。ステップ４１５において、ｍｓｇｉｄが追加のローカリティに存在する場合、プロセスはｍｓｇｉｄを、ステップ４３０にて、ネットワーク６５８を介して翻訳エンジン、例えばＧｏｏｇｌｅ　Ｔｒａｎ

ｓｌａｔｅ ＡＰＩに要求を送信し、翻訳された文字列を受信する。そうでない場合は、プロセスは追加のローカリティ４２０をｍｓｇｉｄで延伸させ（すなわち、プロセスは必要に応じてｍｓｇｉｄを追加ローカリティに追加し）、ステップ４１０に進む。次に、ステップ４４０では、プロセスは翻訳された文字列をメッセージファイルに保存し、そのメッセージファイルは各追加ロケールに、及びデータベース内（例えば、データベース６５６）に提供され、レポートを送信する。ステップ４５０では、プロセスは．ｐｏメッセージファイルのファイルの終端（ＥＯＦ）に到達しているかを確認する。ＥＯＦに到達している場合、プロセスはステップ４６０で終了する。そうでなければ、プロセスはステップ４１０に戻る。

【００４２】

ここで図５を参照すると、例示的な実施形態にかかる、．ｐｏメッセージファイルとＪＳＯＮとの間の行のバリデーションの方法のフローチャートが示されている。ステップ５０５では、プロセスは、ネイティブローカリティについて、．ｐｏメッセージファイルからｍｓｇｉｄを取得する。ステップ５１０では、プロセスは全ての存在するローカリティについて、ローカリティ参照ファイル（すなわち、ＪＳＯＮ ｉ１８ｎファイル）を構文解析する。ステップ５２０では、プロセスは．ｐｏメッセージファイル内のキー／値が、デフォルトで値が空であるネイティブロケールを除き、ローカリティ参照ファイル（すなわち、ＪＳＯＮ ｉ１８ｎファイル）と等しいかを判定する。そのため、ネイティブロケールについてはキーの値についてのみ比較する。．ｐｏメッセージファイル内のキー／値がローカリティ参照ファイルと等しい場合、プロセスは、ステップ５３０でＪＳＯＮ（ローカリティ参照ファイル）ファイルがファイルの終端（ＥＯＦ）に到達しているかを判定する。ＪＳＯＮファイルがファイルの終端（ＥＯＦ）に到達している場合、プロセスはステップ５６０で終了する。そうでない場合は、プロセスはステップ５１０に進む。ステップ５２０で．ｐｏメッセージファイルがローカリティ参照ファイルと等しくない場合、プロセスは不在の文字列をデータベース６５６に保存し、ステップ５４０で責任関係者（例えば、担当テクニカルライター）にレポートを送信し、ステップ５５０に進む。ステップ５５０では、エラーは手動で処理される。

【００４３】

図６を参照すると、概して６５０での、地域化されていない行の自動検出、修正、及び翻訳についてのシステム並びに構成要素が示されている。この説明は、コンピュータ、又はコンピュータのネットワークで実行される、プログラム、データ構造又は手順の観点から提示される。システムに実装されたソフトウェアプログラムは、解釈された、コンパイルされた、又はその他の任意のプログラム言語より記述されていてもよい。これらの言語としては、これらに限定されないが、ＰＨＰ、ＡＳＰ．ｎｅｔ、ＨＴＭＬ、ＨＴＭＬ５、Ｒｕｂｙ、Ｐｅｒｌ、Ｊａｖａ、Ｐｙｔｈｏｎ、Ｃ＋＋、Ｃ＃、ＪａｖａＳｃｒｉｐｔ（登録商標）、及び／又はＧｏプログラミング言語を含んでもよい。当業者であれば当然に、代わりに、又は上述と組み合わせて他の言語も利用され得ることを理解すると認識され、例えば、Ｒｕｂｙ ｏｎ Ｒａｉｌｓ、Ｎｏｄｅ．ｊｓ、Ｚｅｎｄ、Ｓｙｍｆｏｎｙ、Ｒｅｖｅｌ、Ｄｊａｎｇｏ、Ｓｔｒｕｔｓ、Ｓｐｒｉｎｇ、Ｐｌａｙ、Ｊｏ、Ｔｗｉｔｔｅｒ Ｂｏｏｔｓｔｒａｐなど、ウェブ及び／又はモバイルアプリケーションフレームワークをも使用できることを認識されたい。更に、本明細書に開示されるシステム及び方法は、例えばインターネットなどのコンピュータネットワーク上で利用可能な、サービスとしてのソフトウェアに実装されてもよいように認識されたい。更に、本開示は、１つ以上のアプリケーションプログラミングインターフェース又はその他を介して、ウェブサービス、アプリケーションプログラミングインターフェース及び／又はサービス指向アーキテクチャを有効としてもよい。

【００４４】

図６は、地域化されていない行の自動検出、修正、及び翻訳のためのシステムを示す。本開示の少なくとも１つの実施形態では、システムは、ユーザＧＵＩ６５２、サーバ６５４、データベース６５６、及びコンピュータネットワーク６５８を備えている。

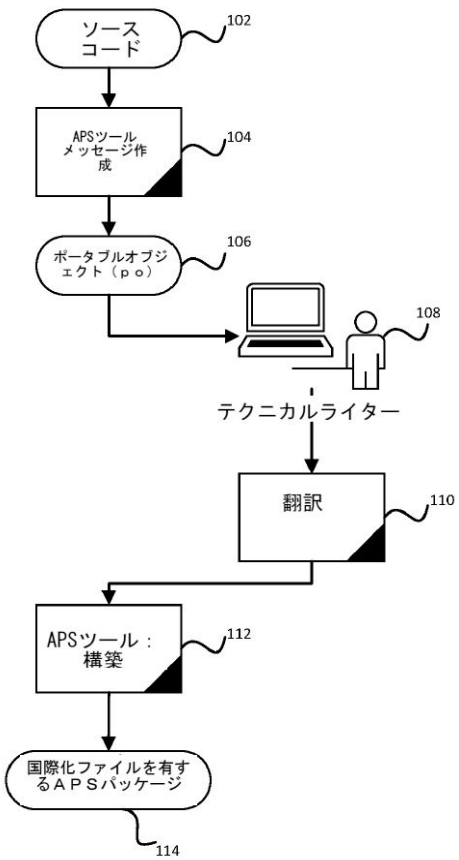
【００４５】
ユーザＧＵＩ６５２は、コンピュータネットワーク６５８を介して、サーバ６５４上に格納されたウェブサービス及び／又はアプリケーションプログラミングインターフェースインフラストラクチャに情報を送信し、並びにそれと概して対話するように構成されてもよい。ユーザＧＵＩ６５２は、サーバ６５４上のウェブサービスインフラストラクチャとの通信がコンピュータネットワーク６５８を介して可能になるよう、ウェブブラウザ、モバイルアプリケーション、ソケット若しくはトンネル、又は他のネットワーク接続ソフトウェアを含んでもよい。
【００４６】
ユーザＧＵＩ６５２は、１つ以上のコンピュータ、スマートフォン、タブレット、ウェアラブル技術、コンピューティングデバイス、又は従来技術で周知の種類のシステム、例えばメインフレームコンピュータ、ワークステーション、パーソナルコンピュータ、ラップトップコンピュータ、ハンドヘルドコンピュータ、セルラー電話、ＭＰ３プレーヤ、若しくはパーソナルデジタルアシスタントを含む。ユーザＧＵＩ６５２は、当業者が思い浮かべるようなソフトウェア、ハードウェア、及び構成要素、例えば１つ以上のマイクロプロセッサ、メモリシステム、入出力デバイス、デバイスコントローラなどを備える。ユーザＧＵＩ６５２はまた、音声若しくは音量調整、（マウスなどの）ポインティングデバイス、キーボード、タッチスクリーン、マイクロフォン、音声認識、及び／又は従来知られる他のデータエントリ手段など、ユーザＧＵＩ６５２の顧客によって操作可能なデータエントリのための１つ以上のデータエントリ手段（図６では示さず)をも備えている。ユーザＧＵＩ６５２はまた、顧客が知覚可能な様態で情報が表示され得る、液晶ダイオードディスプレイ、発光ダイオードディスプレイなどの既知の様々な種類を含み得るディスプレイ手段をも備える。なお、ユーザＧＵＩ６５２は、本開示に従ってユーザＧＵＩ６５２に割り当てられた機能を動作可能に実行するために、当業者が思い浮かべるようなソフトウェア、ハードウェア、及び構成要素を更に備えてもよいことを認識されたい。
【００４７】
データベース６５６は、システムによって生成された、及び／又は１つ以上の情報源から取得された情報を記憶するように構成される。本開示の少なくとも１つの実施形態では、データベース６５６がサーバ６５４上にあるように、データベース６５６をサーバ６５４と「関連付ける」ことができる。データベース６５６はまた、データベース６５６がサーバ６５４から離隔したサーバ又はコンピューティングデバイス上にあるようにサーバ６５４と「関連付ける」こともできるが、その場合は、その離隔したサーバ又はコンピューティングデバイスが、例えば、Ａｍａｚｏｎ　ＡＷＳ、Ｒａｃｋｓｐａｃｅ、若しくは他の仮想インフラストラクチャ、又は任意のビジネスネットワークなど、サーバ６５４と双方向のデータ転送が可能であることが条件である。本開示の少なくとも１つの実施形態では、データベース６５６が存在する離隔したサーバ又はコンピューティングデバイスは、その離隔したサーバ又はコンピューティングデバイスがサーバ６５４と連続的な双方向データ転送が可能となるよう、サーバ６５４と電気的に接続されている。
【００４８】
明確性のために、図６にデータベース６５６を示しており、本明細書では単一のデータベースとして参照される。当業者であれば、データベース６５６は従来技術で周知の種類のソフトウェアシステムによって接続される複数のデータベースを備えることができ、そのソフトウェアシステムが本開示によるデータベース６５６に委任された機能を実行するために協働して動作可能であることを認識されたい。データベース６５６はまた、大きなデータサービスについては、例えば、Ｈａｄｏｏｐアーキテクチャなどの分配されたデータアーキテクチャの一部であってもよい。データベース６５６は、リレーショナルデータベースアーキテクチャ、ｎｏＳＱＬ、ＯＬＡＰ、又は他の既知の種類のデータベース技術を備えてもよい。データベース６５６は、例えば、マイクロソフト社のＳＱＬサーバ、マイクロソフト社のＡＣＣＥＳＳ、ＭｏｎｇｏＤＢ、Ｒｅｄｉｓ、Ｈａｄｏｏｐ、又はＩＢＭ社のＤＢ２データベース管理システム、又はＯＲＡＣＬＥ社若しくはＳＹＢＡＳＥ社よ

り入手可能なデータベース管理システムなど、多くの周知のデータベース管理システムのうちの１つを備えてもよい。データベース６５６は、ユーザＧＵＩ６５２又はサーバ６５４からデータベース６５６に通信された情報を、取出し可能に記憶する。

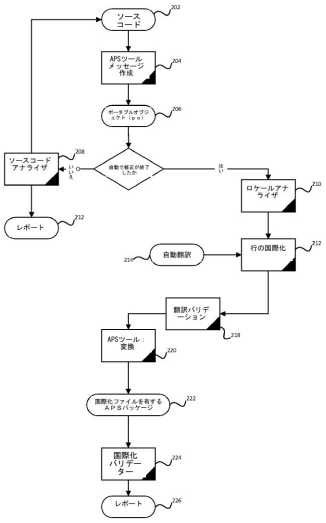
【００４９】

このように、ある実施形態を説明したことで、当業者であれば、記載の方法及びシステムの特定の利点が達成できていることが明らかに分かるであろう。また、様々な変形、適合、及び代替実施形態が、本開示の範囲及び趣旨の中でされ得ることも理解されたい。開示は、以下の請求項によって更に規定される。

【図１】



【図２】



【図３】

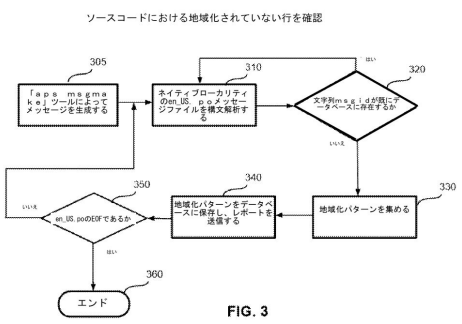


FIG. 3

【図４】

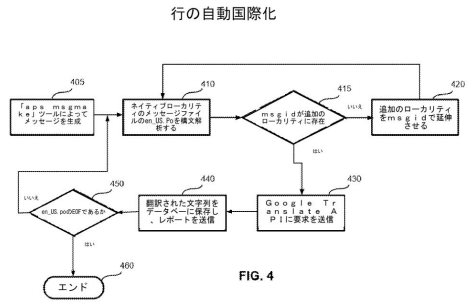
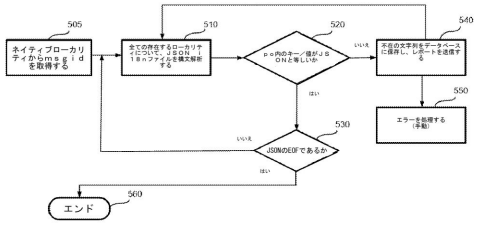


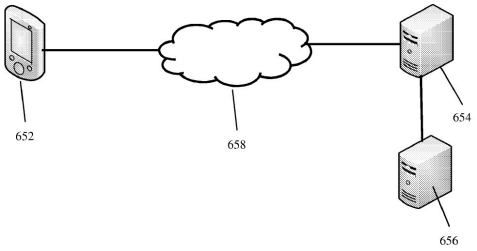
FIG. 4

【図５】

メッセージファイルとＪＳＯＮとの間の行のバリデーション



【図６】



フロントページの続き

(56)参考文献 特開２０００－２０７３９９（ＪＰ，Ａ）
特開２００１－１２５７９４（ＪＰ，Ａ）
米国特許出願公開第２０１６／００９８２６１（ＵＳ，Ａ１）
太田 一樹，メッセージの仕組みと作成法，ＵＮＩＸ ＵＳＥＲ，ソフトバンクパブリッシング株式会社，２００５年 4月 1日，第１４巻，第４号，p.78-84

(58)調査した分野(Int.Cl.，ＤＢ名)
Ｇ０６Ｆ 8／７０－8／７７