Common Locale Information Repository Project

日本アイビーエム株式会社
ソフトウェア研究所

野地　健太郎

Abstract: Common Locale Information Repository Project は、Linux, Java, 商用 UNIX などで実装されているロケールを、オープンで交換可能なデータフォーマットとして記述し、共用のレポジトリを開発することを目的としている。そのために、ＸＭＬを使ってロケール・データを記述し、オープンソースでレポジトリを開発する予定である。この論文では、ロケールの実装の現状、問題点を論じ、このプロジェクトの概要、現在検討中の XML 形式の Locale Data や技術的方向性を論じる。

1. はじめに

ロケールは、ソフトウェアの国際化モデルのベースとなる概念であり、通常、言語及び地域依存情報のことを指す。例えば、日付の形式や数字の表現形式、あるいは貨幣記号などは、言語や地域に依存しているので、ロケールとして扱うことができる。一般的に、ロケールは言語、地域、Modifier を組み合わせた識別子を提供し、ロケールを切り替えることによって各国語をサポートしている。同言語かつ同地域のロケールであれば、そのロケールに定義されたデータも一致していると想定しがちだが、実際のロケールを調査してみると、プラットフォームやベンダーによって少しずつ異なり、必ずしも一致していない。その結果、クロス・プラットフォームの分散環境では、ロケールの互換性、整合性は、保障されない。本論文では、現状のロケールに関する検証、現状分析と問題点を論じ、次に解決策として Common Locale Information Repository Project について論じる。

2. 現状分析

2.1. ロケール

国際化プログラミング[1]によるとロケールは、ANSI C で各国語をサポートするために導入された概念であり、国際化アプリケーション実行モデルの根本になるものである。ロケールは、その後 POSIX 規格などにも受け継げられ、Linux や Java においてもソフトウェア国際化の基本となっている。しかしながら、ロケールの定義自体は、現在においても曖昧であり、言語や地域に依存した情報データベースを指すこともあれば、単に言語地域を識別するための ID であることもある。ここでは、現在一般的に実装されているロケールについて現状を示す。

2.2. POSIX ロケール

POSIX におけるロケールは厳密に定義されているわけではなく、ロケールによって、関数の動作が変

わるということだけが決められている。このようなロケールに依存する関数は、地域や言語、習慣によって動作が変わる。POSIX では、ロケール依存関数のために、以下の 6つのロケール ・カテゴリが用意されている。

LC_TIME 日付や時間表記は、言語や地域によって異なる。例えば、02/08/05 という表記が、2002 年の 8月 5日のことか、2005 年の 2月 8日のことかは地域によって変わる。ここでは、このようなデータの表現形式に関する規定や、月、曜日に関するデータなどが定義される。

LC_MONETARY 通貨単位や数字の区切り桁数、区切りの記号などは地域によって異なる。ここでは、金額表示金額表示の表現形式や、通貨単位が定義される。

LC_NUMERIC 小数点の記号や符号の位置、符号記号は、言語や地域によって異なる。ここでは、数値表記に関する表記形式や正負記号などが定義される。

LC_MESSAGES はい、いいえ、などの応答メッセージは言語によって異なる。ここでは、応答メッセージが文字列や正規表現で定義される。

LC_COLLATE コレーションは、言語や地域によって異なる。ここでは、コレーションの順序が定義される。

LC_CTYPE 文字種(大文字、小文字など)の定義や分類は、言語によって異なる。ここでは、文字種が定義される。

POSIX は、ロケールのカテゴリとロケールによって動作を変えるべき関数やロケール ・データ表記のシンタックスを規定したが、ロケール依存情報を格納するロケール ・データベースの実装や、ロケールのデータに関する標準化は何も決めていない。POSIX 仕様に則った商用の UNIX システムは、各ベンダーがロケール ・データベースを実装し、ロケール ・データを定義している。

2.3. Linux ロケール

Linux のロケールは、GNU GlibC が提供している。GlibC のロケールは、一部 POSIX ロケールを拡張しているため、POSIX のカテゴリに加えて、以下のロケール ・カテゴリを定義している。

LC_PAPER 通常使用するペーパーサイズの仕様は、地域によって異なる。ここでは、ペーパーサイズの幅、高さなどのデータが定義される。

LC_ADDRESS 住所の表記は、言語、地域によってことなる。ここでは、住所の要素、並びが定義される。

LC_NAME　　姓名は、国、言語によって構成要素、並びが異なる。ここでは、名前の構成要素、並び、タイトルが定義される。

LC_TELEPHONE　電話番号は、国によって表記法が異なる。ここでは、表記形式が定義される。

LC_MEASUREMENT　単位系は、国によって異なる。ここでは、単位系を示すIDが定義される。

以上のカテゴリは、GNU GlibCのみサポートされており、通常のPOSIXでは利用できない。

2.4. ICUロケール

ICU- International Components for Unicode[2]は、Open source Projectで開発されているUnicodeを利用するためのコンポーネントである。C 言語のためのライブラリー、C++言語のためのクラス、Java言語のためのクラスが提供されており、Windows, UNIX, Linux など様々なプラットフォームで利用できる。ICUは、ロケールやUnicodeを扱うための機能がAll in Oneで提供され、Javaの国際化フレームワークの基礎となっている。ICUロケールは、以下2.5に述べるJavaロケールと同様の機能を提供する。

2.5. Javaロケール

Javaのロケールは、POSIXやLinuxとは、異なったフレームワークを提供している。その理由として、Javaが、Unicodeをベースに国際化を採用したこと、クロス・プラットフォームをサポートしていること、オブジェクト指向であること、があげられる。Java のロケールは、オブジェクトであり、その構成要素は、言語と(国)地域である。Java の幾つかのクラスは、ロケール・オブジェクトの影響を受けて動作が変化する。例えば、java.text.NumberFormat は、ロケール・オブジェクトを引数にすることで、ロケールにあわせた数値フォーマットを行う。これらのクラスの中には、通貨単位や曜日、日付などを処理するものがあり、そのためにJavaロケールのリソースには、以下のようなロケールに付随するデータが定義されている。

- 時刻、日付に関するリソース
- 数値に関するリソース
- 金額に関するリソース
- コレーションに関するリソース
- その他、言語IDなど

Javaロケールは、詳細な仕様は公開されておらず、またJavaの実行環境に内包されているため、ユーザーがリソースを変更することはできない。ただし、Java環境であれば、どのベンダーのJava環境でも、ロケールの整合性は保証されている。

2.6. MS Windows ロケール

MS Windows のロケールは、例えば Windows 2000 の日本語版では、コントロールパネルの地域と言語のオプションで定義されている。ここでは、ロケール情報として、以下のカテゴリが用意されている。

数値 数値表記における桁の区切り、桁区切りの記号、小数点の記号、符号、単位などが定義される。
通貨 通貨表記における通貨記号、符号、桁の区切り、桁区切りの記号、小数点の記号など。
時刻 時刻の形式、午前、午後など。
日付 日付の短い形式、長い形式、カレンダー
入力 入力言語

Windows ロケールのデータは、ユーザーが、コントロールパネルの地域と言語のオプションで、変更できる。コレーションは、Windows ロケールの一部としてではなく、MS SQL Server の一部として実装されている。

3. 現状の問題点

以上、代表的な既存ロケールを概観したが、これらのロケールに共通することは、以下の 3つである。

1) 言語や国（地域）を指し示す識別子の存在
2) ロケールに付属するカテゴリの存在
3) ロケール・データを格納するデータベースやレポジトリまたはリソースの存在

一方、ロケールの実装は実装者依存なので、ロケール ID のネーミングや、カテゴリの種類、ロケールのデータなどは、プラットフォームによって異なっている。通常このような実装者依存の部分は、単一システム内で、ロケールを利用している限り問題にならない。それは、閉じたシステム内では、ロケールの互換性、整合性が保障されているからである。しかし、クロス・プラットフォームな分散環境では、この差異部分が問題となる。例えば、Linux の日本語のロケール ID が ja_JP だと仮定する。したがって、この Linux では、どのプロセスにおいても日本語のロケールは、ja_JP である。例えば、ja_JP で動いている、あるプロセスが、異なるプロセスに日本語ロケール ID ja_JP を渡したとしても、そのプロセスは適切に認識できる。次に、分散環境下で、Linux のこのプロセスが Linux 以外のプラットフォーム B で動いているプロセスに、日本語ロケール ja_JP を渡したとする。もしプラットフォーム B の日本語のロケール ID が japanese であった場合、プラットフォーム B は、正しくロケール ID 認識できない。同様の現象が、各ロケール・カテゴリやデータにおいても成り立つ。問題点は、以下の 3つにまとめられる。

a) 分散環境下では、ロケール識別子には、互換性、整合性がない。
b) 分散環境下では、ロケール・データやロケール・カテゴリは、互換性がない。
c) 分散環境下では、ロケール・データを格納するデータ・ベースやレポジトリが共有できない。

ヘテロジニアスな分散環境では、以上の問題が、顕著になる。そこで、この問題を解決するために、プラットフォーム非依存で、ロケール・データの整合性を保つことができる、新しいロケールの仕組みが必要となる。Common Locale Repository Project は、これを実現することを目的としている。

4. Common Locale Information Repository Project

Common Locale Repository Project の目的は、分散環境下でロケール・データの整合性を失わないような仕組みを提供することである。そのためには、

1) プラットフォームに依存しないロケールデータの記述方法
2) クロス・プラットフォーム(オペレーティング・システムやオフィス・アプリケーション)の要求を満たすロケール・カテゴリ
3) ロケールを交換できるメカニズム
4) ロケール・データを共通に格納できるレポジトリ

が必要である。

4.1. ロケール・データの記述方法

プラットフォームに依存しないロケール・データを記述するために、記述言語として XML を利用する。以下は、現在検討中のロケールの記述例である。

```
<?xml version="1.0" encoding="UTF-8" ?>
<localeData>
  <versioning>
    <version number="1.1">Various notes and changes in version 1.1</version>
    <version number="1.0">Various notes and changes in version 1.0</version>
  </versioning>
  <identity>
    <language id="en" />
    <country id="US" />
    <variant id="B"/>
    <correspondsTo vendor="windows">0409</correspondsTo>
  </identity>
  <orientation lines="top-to-bottom" characters="left-to-right" />
  <encodings requiredLetters="a-z&#xE5;&#xE6;&#xF8;">
   <mapping>windows-1252</mapping>
  /encodings>
  <displayNames>
  <scripts>
    <script id="Latin">Romana</script>
  </scripts>
  <languages>
   <language id="ab">Abkhazian</language>
   <language id="aa">Afar</language>
  </languages>
  <countries>
   <country id="AF">Afghanistan</country>
   <country id="AL">Albania</country>
```

```
 </countries>
 <localePatterns>
  <locale>{0,choice,0#|1#{1}|2#{1} ({2})}</locale>
  <list>{0,choice,0#|1#{1}|2#{1},{2}|3#{1},{2},{3}}</list>
  <composition>{0},{1}</composition>
 </localePatterns>
</displayNames>
<calendars>
 <calendar class="Gregorian" default="true">
  <monthNames>
   <month id="1">January</month>
   <month id="2">February</month>
   <month id="3">March</month>
   <month id="4">April</month>
   <month id="5">May</month>
   <month id="6">June</month>
   <month id="7">July</month>
   <month id="8">August</month>
   <month id="9">September</month>
   <month id="10">October</month>
   <month id="11">November</month>
   <month id="12">December</month>
  </monthNames>
  <monthAbbr>
   <month id="1">Jan</month>
   <month id="2">Feb</month>
   <month id="3">Mar</month>
   <month id="4">Apr</month>
   <month id="5">May</month>
   <month id="6">Jun</month>
   <month id="7">Jul</month>
   <month id="8">Aug</month>
   <month id="9">Sep</month>
   <month id="10">Oct</month>
   <month id="11">Nov</month>
   <month id="12">Dec</month>
  </monthAbbr>
  <dayNames>
   <day id="SUN">Sunday</day>
   <day id="MON">Monday</day>
   <day id="TUE">Tuesday</day>
   <day id="WED">Wednesday</day>
   <day id="THU">Thursday</day>
   <day id="FRI">Friday</day>
   <day id="SAT">Saturday</day>
  </dayNames>
  <dayAbbr>
   <day id="SUN">Sun</day>
   <day id="MON">Mon</day>
   <day id="TUE">Tue</day>
   <day id="WED">Wed</day>
   <day id="THU">Thu</day>
   <day id="FRI">Fri</day>
   <day id="SAT">Sat</day>
  </dayAbbr>
  <week>
   <minDays>1</minDays>
   <firstDay>SUN</firstDay>
   <weekendStart><day>FRI</day><time>18:00</time></weekendStart>
   <weekendEnd><day>SUN</day><time>18:00</time></weekendEnd>
    </week>
    <am>AM</am>
    <pm>PM</pm>
    <eras>
```

```
      <era id="0">BC</era>
      <era id="1">AD</era>
     </eras>
     <patterns>
     <chars>GyMdkHmsSEDFwWahKz</chars>
     <time>
      <full>h:mm:ss a z</full>
      <long>h:mm:ss a z</long>
      <medium>h:mm:ss a</medium>
      <short>h:mm a</short>
     </time>
     <date>
      <full>EEEE, MMMM d, yyyy</full>
      <long>MMMM d, yyyy</long>
      <medium>MMM d, yyyy</medium>
      <short>M/d/yy</short>
      </date>
      <dateTime>{0} {1}</dateTime>
     </patterns>
    </calendar>
   </calendars>
   <numberFormat class="decimal" default="true">
       <patterns>
        <decimal>#,##0.###;-#,##0.###</decimal>
        <percent>#,##0%</percent>
        <scientific>0E##</scientific>
    </patterns>
      <symbols>
         <decimal>.</decimal>
         <group>,</group>
         <list>;</list>
         <percentSign>%</percentSign>
         <nativeZeroDigit>0</nativeZeroDigit>
         <patternDigit>#</patternDigit>
         <plusSign>+</plusSign>
         <minusSign>-</minusSign>
         <exponential>E</exponential>
         <perMille>&#x2030;</perMille>
         <infinity>&#x221e;</infinity>
         <nan>&#x2639;</nan>
       </symbols>
    </numberFormat>
    <currencies>
        <currency id="USD" default="true">
           <symbol>$</symbol>
           <pattern>、#,##0.00;(、#,##0.00)</pattern>
           <decimal>.</decimal>
           <group>,</group>
        </currency>
    </currencies>
</localeData>
```

XML を利用するメリットは、XML の処理系を使ってどのプラットフォームでもロケール・データの処理が可能になること、及び図 1のように XML を介在することにより、各プラットフォーム間で、共通のロケール・データを流通させることが可能となることである。ただし、日付のフォーマット・データの記述方法のように、プラットフォームによって、データ記述のシンタックスが異なる場合も多い。したがって、XML ロケールは、どのプラットフォームのシンタックスにも変換が可能な記述能力が必要とされる。第 18回国際ユニコード会議の Universal Locale Project for Linux[3]では、この XML 形式のロケールからLinux 用のロケールを生成し、Linux に対してオープンソースとして提供したことが紹介されている。この論文

では、現在検討中の XML 形式のフォーマットから POSIX, Java, ICU, Linux のフォーマットに変換が可能であることが示されている。

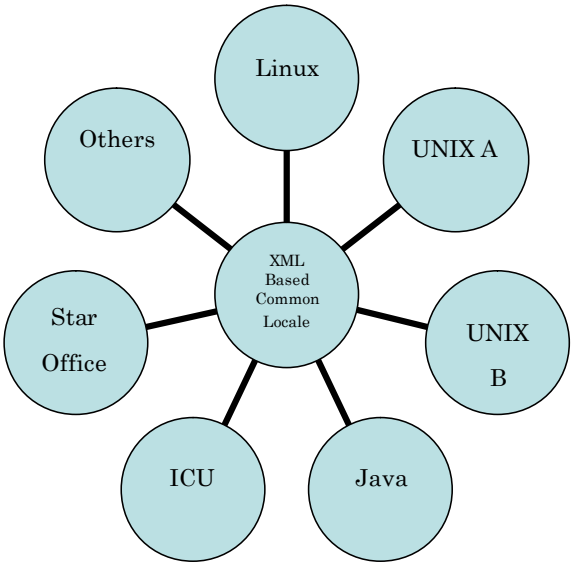

図 1 XML ロケールと各プラットフォーム

4.2. ロケール・データのカテゴリ

ロケールのカテゴリは、プラットフォームによって様々であることは、既に述べたとおりであるが、一部のカテゴリは、どのプラットフォームにも共通して存在している。以下が、前述のプラットフォームにおける共通のカテゴリである。

- 日付の整形、データ
- 時刻の整形、データ
- 通貨の整形、データ
- 数値の整形、データ

一方、以下のカテゴリは、一部のプラットフォームで定義されているカテゴリである。

表 1 一部のプラットフォームで、定義されているロケール・カテゴリ

	Linux	POSIX	Java/ICU	Windows
コレーション	Yes	Yes	Yes	No. (SQL Server)
文字クラス	Yes	Yes	No	No
応答メッセージ	Yes	Yes	No	No
アドレスフォーマット	Yes	No	No	No
ネーム・フォーマット	Yes	No	No	No
ペーパーサイズ	Yes	No	No	No
カレンダー	No	No	Yes	Yes
電話番号フォーマット	Yes	No	No	No
単位系	Yes	No	No	Yes
言語識別子	Yes	Yes	Yes	Yes

これらのカテゴリが三者三様であるのは、国際化のアーキテクチャ、適用範囲が異なっているからである。例えば、コレーションは、通常データベースやオフィスアプリケーションで多用するので、システム層よりもデータベース層や、アプリケーション層でサポートすることが適切な場合もある。文字クラスは、Unicode に依存する場合と、Locale に依存場合がある。Common Locale Information Repository は、異なるプラットフォーム間で交換可能であることを目的にしているので、以上のすべてのロケールを網羅できるようにカテゴリを定義する方針である。

4.3. 分散環境下におけるロケール・データの交換

ロケール・データを、クロス・プラットフォームで交換可能にするためには、プラットフォームに依存しないXMLを用いることで可能であるが、考慮すべき方針として、以下の3つがある。

- ロケール・データ交換のプロトコル

XMLで記述されたロケール・データは、HTTP, FTPなどの既存のネットワークプロトコルにのせてデータの交換を可能とし、特定のプロトコルには依存しない。また、SOAP を使って、SOAP エンベロープとしてデータの送受信が可能になるようにスキーマを定義する。

- ロケール・データのモジュール化

ネットワークでの送受信やクロスプラットフォーム・サポートを適切に行うためには、ロケール・カテゴリ別に交換できなくてはならない。前述の例のように、ロケールは、calendarやnumberなどのカテゴリ別にデータ交換が可能となるようにロケール・データのモジュール化を行う。

- 交換を行う上での双方のシステムが有している情報の同一性の検証

ロケール・データを交換する場合、ロケールをリクエストしたリクエスターのロケールとリクエストされた側のロケールが同一かどうか、検証する必要がある。もし、両方のシステムに同じロケールがあれば、交換する必要はないからである。同一性の検証には、システムに存在しているロケールのレジストリを用意する必要がある。レジストリには、インストールされているロケールとそのカテゴリやidentificationが登録される。レジストリを比較することによって、システムに存在しているロケールの同一性が、検証可能となる。

4.4. ロケールデータのバリエーションと識別子

以下は、幾つかのプラットフォームの日本語ロケールの日付のサンプルである。

表2 日本語ロケールの日付の例

Linux	2002年8月6日
Windows	短い形式: 2002/08/06 長い形式: 2002年8月6日
Java	FULL: 2002年8月6日 LONG: 2002/08/06 MID: 2002/08/06 SHORT: 02/08/06
AIX	Mon Aug 6
その他	平成14年8月6日

この例から明らかのように、日本語の日付の形式といっても多くのバリエーションがある。それは、日

本語の日付形式のようなデータは、アプリケーションやシステムの要求よって様々であり、これを単純に1つに決めてしまうことはできないからである。このような変形を許し、区別できるようにするためには、ロケール識別子として、言語、地域に加えてこれらの変形を区別できるための Modifier を導入する必要がある。以下がその例である。

```
<identity>
       <language id="ja" />
       <country id="JP" />
       <variant id="B">
       <correspondsTo vendor="Linux">####</correspondsTo>
       </variant>
</identity>
<date>
       <full>yyyy 年 MMMM 月 d 日</full>
</date>
```

この例では、日本語、日本国、Linux のロケールの日付形式を定義していることがわかる。

4.5. レポジトリ

ロケール・レポジトリは、各ロケール・データが格納される。ロケールは、XML で記述されているので、データは、既に構造化、階層化されている。したがって、ロケール・レポジトリの階層構造は、XML の定義と同一となる。現在検討中のレポジトリの論理的な構造は、以下のように階層化される。

```
+ en
   + US
     + linux
       + localeData
       + versioning
       + identify
       + orientation
       + encodings
       + displayNames
       + calendars
       + numberFormat
       + currencies
     + windows
     + java
     + icu
   + GB
   + CA
```

現時点では、レポジトリの物理的な実装は、規定していない。

4.6. ロケールデータの公開、登録

ロケール情報は、既にオープンソースで存在しているリソースや、このプロジェクトのメンバーからデータを集める。既に、ICU のグループは、XML をベースにしたロケールをオープンソースとして公開している。レポジトリの開発や管理はオープンソースで行う方針である。

5. 今後の課題

XML を使用することにより、プラットフォームに依存しないロケールのデータ記述は可能となるが、幾つかの課題も残されている。

1) セキュリテイー

ロケールを交換することによって、送られてきたロケールを受け入れるかどうが、という問題が発生する。これを解決するためには、ロケールの信用性を保証する仕組みが必要となる。現在は、Locale の Identification を利用することを検討中である。

2) ロケール・データの管理、評価

ロケール・レポジトリの信頼性を保つためには、登録されたロケールの管理方法、登録する場合の評価方法を確立する必要があり検討中である。

6. まとめ

ロケールは、ソフトウェアの国際化の基本概念であるが、その定義はプラットフォームによって異なっていた。その結果、ロケールの分散環境における互換性の欠如が問題となっていたが、XML を使ったロケールの記述方法と、交換方法、ロケール・レポジトリの導入によって、プラットフォーム間のロケール・データの交換が可能となり、ロケール・データの互換性が可能となる。また、ロケールは、共通の資産となるデータであり、共有できるリポジトリによって、ベンダーにとっては管理コストの減少につながる。現在、XML のロケール記述の評価、レポジトリの開発を行っている。

謝辞

ICU 開発グループの Helena Chapman 氏には、XML ロケールのデータの提供を始め、有益なご助言をいただき、ありがとうございました。Li18nux の木戸　彰夫氏、樋浦　秀樹氏には、技術的な問題点や解決策などご助言をいただき、ありがとうございました。

参考

[1] **清兼　義弘、末廣 陽一/編著：国際化プログラミング i18n ハンドブック, 共立出版**

[2] ICU – International Components for Unicode, http://oss.software.ibm.com/icu/

[3] Kentaorh Noji, Tetsuji Orita,：Open source project, Unicersal Locales for Linux, http://www.unicode.org/iuc/iuc18/papers/a2-presentation.pdf