オペレーティング・システム 33－1
（1986. 12. 12）

オペレーティング・システムの多国語サポート

大平 剛　　大場 充

日本アイ・ビー・エム株式会社　　東京基礎研究所

本論では、オペレーティング・システムで、多国語をサポートする方法について述べる。多国語をサポートするためには、2 バイト文字を扱う必要がある。我々は単一2 バイトの文字セットを採用した。これは既存のオペレーティング・システムのプログラム・コードを、再利用することを考えての事である。また拡張性や再利用を考えた場合の、オペレーティング・システムの構成についても述べる。

現在のプロトタイプは、日本語と英語をサポートしている。コマンド、システム・メッセージは、システム内では中間言語で扱われ、利用者は、ファイル名、ユーザ名、コマンド、システム・メッセージを母国語文字で使用できる。

Multi Languages Support to an Operating System

Tsuyoshi OHIRA　Mitsuru OHBA

Tokyo Research Laboratory, IBM Japan, Ltd
5-19 Sanban-cho, Chiyoda-ku, Tokyo 102, JAPAN

The paper describes a way of adapting an existing operating system to user's (national) language that requires the handling of a large character set (Japanese, Chinese, Korean, etc.).

A prototype system (essentially a Japanese version of the IBM VM/370) was implemented on the basis of the architecture that handles uniform two-byte codes. Commands and system messages were changed to intermediate representations for supporting of multi-lingual characters. Users can use userids, fileids, commands, and system messages which consist of native characters.

The prototype was implemented considering reusability for multi-byte codes. The reuse of old code (the VM/370) was stimulated in developing the prototype. The design of the prototype suggests a general approach for future multi-lingual operating systems.

1 はじめに

事務処理のオフィス・オートメーション化が進むにつれて、日本語処理機能の需要が増している。ワード・プロセッサにより、日本語文書の作成や印刷が容易になり、パーソナル・コンピュータでは、システム・メッセージが日本語で出力されるものもある。また日本語をサポートするアプリケーション・ソフトウェアも増えてきた。しかし大型計算機では、日本語のサポートはまだ十分ではない。

日本語化の対象となるものは、ユーザ名、ファイル名、コマンド名、システム・メッセージ等、通常オペレーティング・システムで扱われる文字列である。日本語化を行うためには、2バイト・コードを扱えるようにする必要がある。2バイト・コードをサポートすれば、2バイトで表現される文字は、漢字以外でも使用可能となり、多国語サポートもしやすくなる。

現在2バイト文字セットには、いくつか種類があり、また文字列の処理方法にも大きく分けて2種類ある。これらのうちどれを採用するかで、アプリケーション・ソフトウェアの文字列処理方法が大きく影響を受ける。

多国語サポートを考えた場合、「どのような変更が、システム自体に与える影響を、より小さくできるか。」また「どのような設計方法が、拡張性を大きくできるのか。」というソフトウェア工学上の問題もある。

以下2節では、2バイト文字を適用する際に考えられる、二つの方法について述べる。3節では、現在使用されている2バイト文字セットについて、その種類と特徴を述べる。4節では、多国語をサポートするために有効と思われる方法について、5節では、多国語サポートの実験例について述べる。そして最後にまとめとして、評価やこれから問題となることについて述べる。

2 アーキテクチャ

ユーザ言語に依存しないプログラムの設計、つまり2バイト文字の使用を許す設計には、次の2種類の方法が考えら、それぞれ次のような特徴がある。

1. 混在型(Mixed Mode)
   シフト文字または文字コードのビットを使って、1バイト文字と2バイト文字を区別する。
   - 既存のデータ・ファイルをそのまま使用できる
   - 部分的な変更が容易
   - 文字列処理が複雑になる
2. 単一型(Uniform Mode)
   単一サイズの文字コードを使う。
   - 文字列処理は1バイト文字セットと同様のアルゴリズム
   - 既存のデータ・ファイルは処理時に変換が必要
   - 部分的な変更がしにくい

既存のオペレーティング・システムの変更を対象とした時の、二つの方法を比較する。1では2よりも段階的な変更を考えたとき、初期の投資は小さい。理由は、既存のデータを変更する必要がなく、部分的な変更が容易であるから。しかし多国語サポートを完全に行うための総コストは、1の方が大きくなる。理由は、文字列処理が複雑になり、2の方が変更が容易であり、拡張性が大きいからである。

1では文字列処理において、1バイト文字と2バイト文字を区別するという操作が必要となり、処理を複雑にする。そのため、ソフトウェア工学での信頼性、保守容易性、変更容易性の点で、2の方法が有効と思われる。

3 文字セット

最近では次の4種類の文字セットが使用されている。

1. 単一2バイト型(uniform two byte mode code)
2. 自己定義混在型(self-defined mixed mode code)
3. 非自己定義混在型(non-self-defined mixed mode code)
4. シフト混在型(shifted mixed mode code)

単一2バイト・コードは、1文字のコード長が全て2バイトである。EBCDICの場合は、特殊文字を含む全ての英数字は、第一バイトがパディング・バイトの機能を持つように2バイト拡張されている。

2は最近UNIXユーザ・グループによって選択されたものであり、ASCIIコードを拡張したものである。このコードでは、既存のASCII文字に影響を与えない。拡張のため、文字コードの8ビット目を使用し、第一バイトと第二バイト共に8ビット目を2バイト・コードの識別ビットとして使用する。また第一バイトと第二バイトのコードが重複しないように定義されている。従って全てのコードが自己定義され、1バイト・コードか2バイト・コードであるか、2バイト・コードであれば第一バイトか第二バイトであるかを識別できる。しかし文字の種類が約8800に制限される。

3は2バイト・コードの第一バイトの範囲が、X'81'からX'9F'およびX'E0'からX'FC'である。第一バイトの8ビット目が、2バイト・コードであるかどうかの識別のために使用される。第二バイトは第一バイトや1バイト・コードと重複することがある。従って、与えられた1バイトが、1バイト・コードか2バイト・コードの第一バイトか第二バイトであるかを、識別することができない。

4は特別な制御文字を使用して、1バイト文字と2バイト文字の切り換えを表わす。この制御文字はSO(shift-out),SI(shift-in)と呼ばれ、これによって2バイト・コードが囲まれる。従って、与えられた1バイトが1バイト・コードか2バイト・コードの第一バイトか第二バイトであるかを識別することができない。

上記の4種類の文字セットの、文字列処理における長所短所を比較すると以下のようになる。

1. 単一2バイト型
   - 文字列処理のアルゴリズムに影響を与えない。
   - マッチング・アルゴリズムが単一1バイトのものと同じ
   - 与えられた1バイトが、1バイト文字か2バイト文字かを識別する必要がない
2. 自己定義混在型
   - 文字列処理のアルゴリズムに影響を与えない。
   - マッチング・アルゴリズムが単一1バイトのものと同じ

- 与えられた1バイトが、1バイト文字か2バイト文字のどのバイトか識別できる

3. 非自己定義混在型
   - 文字列処理のアルゴリズムに影響を与えない。
   - マッチング・アルゴリズムが単一1バイトのものと違う
   - 識別するために、2バイト文字の先頭を見つける必要がある
4. シフト混在型
   - 文字列処理のアルゴリズムが単一1バイトのものと違う
   - マッチング・アルゴリズムが単一1バイトのものと違う
   - 識別するために、直前のシフト・コードを見つける必要がある

現在では、ソフトウェアの日本語化を行う場合、2,3,4の方法を取ることが多い。その理由は、部分的な変更をしやすく、既存のデータ・ファイルを変更する必要がないからである。また1バイト文字は半角、2バイト文字は全角というように、現在のディスプレイ装置への表示方式に合っていることによると考えられる。

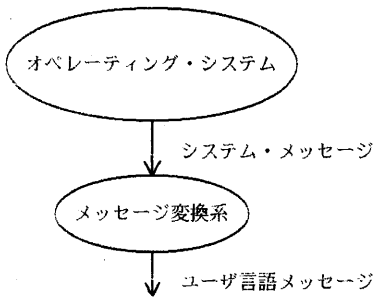

図1．実行時翻訳方式

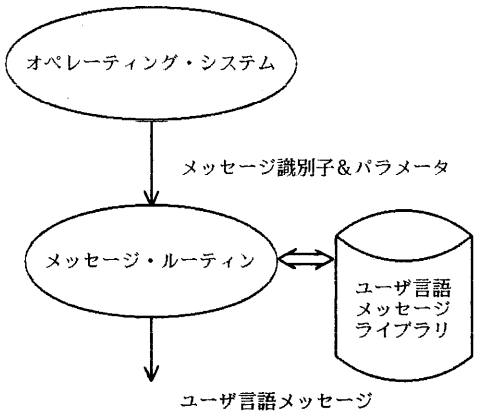

図2．メッセージ・ライブラリ方式

4 多国語サポート

オペレーティング・システム上で多国語化を行う対象となるものは、以下のものである。

1. システム・メッセージ
2. コマンドおよびキーワード
3. ユーザ名
4. ファイル名

これらに対して多国語をサポートするためには、1に関しては、図1,2にあるように、実行時翻訳する方法とメッセージ・ライブラリを使用する方法がある。実行時翻訳する場合、各ユーザ言語ごとに翻訳系を追加して、生成されたメッセージを翻訳系に渡すようにする。メッセージ・ライブラリを使用する場合は、メッセージ・テキスト自身をユーザ言語に翻訳して、ライブラリにまとめる。メッセージ・テキストは、メッセージ・モジュールにまとめられているものと各処理モジュールに分散しているものとがある。

2に関しては、図3に示すように、一般には、コマンド列がコマンド解析ルーチンで解析され、特定のコマンド処理ルーチンが呼び出される。解析ルーチンの変更は、単純な方法では、コマンド表の変更と、キーワードがある場合は、コマンド処理ルーチンのキーワード解析部分を変更する。

3,4に関しては、名前の長さやコード変換部分などの変更を行う。

それ以外では、文字コードのサイズが違うことから、文字列長、文字カウンタ、データ領域等の変更や、それに付随した命令の変更を行う。

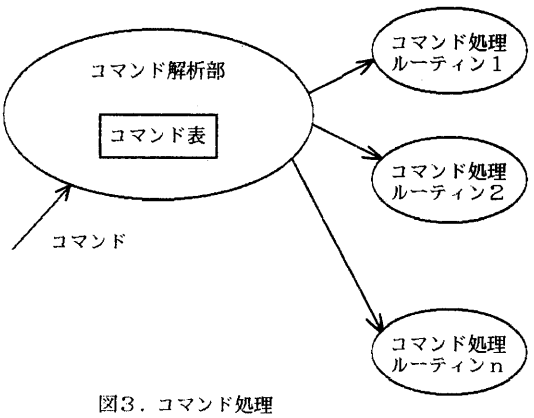

図3．コマンド処理

4.1 コマンド構文解析系

オペレーティング・システムのコマンド解析は、通常コマンド表を引いて、対応するコマンドの処理ルーティンを呼出し、パラメータがある場合は、各コマンド処理ルーティンで、パラメータの解析を行ない対応する処理をする。従ってコマンドの構文解析と意味解析を同時に行っていることが多い。

多国語をサポートするためには、構文解析部と意味解析部を分けることが望ましい。図4に示すように、まずユーザ言語で記述されたコマンドを、ユーザ言語構文解析系が処理し、意味解析を行うための中間言語に変換する。この際オペレーティング・システムの内部では、直接文字列を使用せず、文字コードに依存しないように、トークンを使用することが有効である。つぎにコマンド解析部が、トークン列を解析して、対応するコマンド処理ルーティンを呼び出す。各処理ルーティンの意味解析部は、トークン列を解析して各処理を行う。この場合、ユーザ言語構文解析系はユーザ言語ごとに異なり、意味解析系はユーザ言語には依存せず、共通となる。

コマンド構文解析系は、コマンド解析部と独立しているため、自然言語の構文解析が可能なものが存在すれば、それを使用してもよいし、単にキーワードを変換するだけの単純なものでもよい。

4.2 メッセージ合成系

オペレーティング・システムのシステム・メッセージは、メッセージ・モジュールにまとめられているものと、各処理モジュールに分散しているものとがある。メッセージ・モジュールにまとめられている場合には、そのモジュールを多国語サポート用に変換すればよいが、分散しているものは、各処理モジュールすべてを変更しなければならない。従って、メッセージ・テキストを全て各処理モジュールから抜き出して、メッセージ・モジュールにまとめることが望ましい。

コマンドの各処理部でメッセージを発生したい時には、メッセージ識別子とパラメータを、メッセージ合成系に送る。メッセージ合成系はメッセージ表から登録されている識別子に対応するメッセージ・テキストとパラメータから、合成したユーザ言語のメッセージを出力する。

メッセージの合成方法にも、実行時のメッセージ翻訳とメッセージ・ライブラリの方法があり、現在では、自動翻訳の難しさから、メッセージ・ライブラリの方法を取る場合が多い。

4.3 ユーザ言語に依存しない設計

以上述べてきたように、コマンド構文解析系、メッセージ合成系、辞書などは、ユーザ言語ごとに用意する必要がある。多国語をサポートするための設計は、ユーザ言語に依存した部分を、オペレーティング・システム本体から切り放すことである。こうすることで、オペレーティング・システム本体を変更せずにコマンド構文解析系やメッセージ合成系を、目的と技術的制限に応じて、選択、変更できる。現在コマンド解析系やメッセージ合成系が、キーワード置き換えやメッセージ・ライブラリで実現されることが多いと思われるが、将来自然言語認識や自動翻訳にすばらしい処理系が実現した場合でも、容易に変更が可能である。

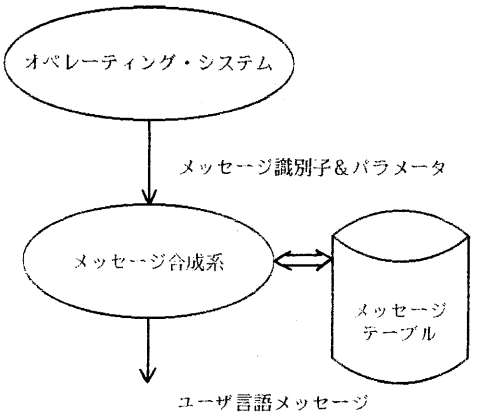

図5．メッセージ合成系

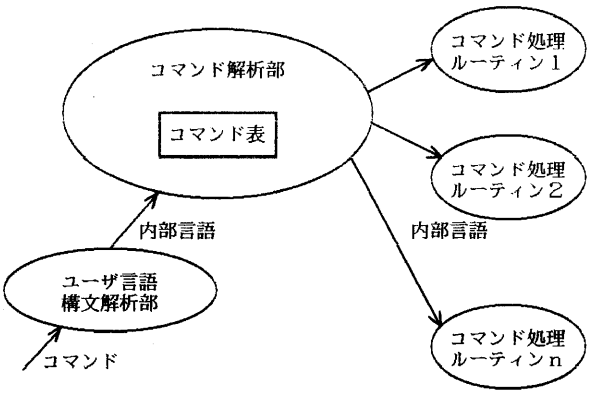

図4．ユーザ言語コマンド処理

5 実験例

IBMのオペレーティング・システムであるVM/CP, VM/CMSに対して、多国語サポートのための変更を部分的に行った。この時の文字セットとしては、単一2バイト型EBCDICを使用した。変更項目は以下である。

1. エラー・メッセージを内部言語方式で実現
2. その他のメッセージは文字列の切り換え
3. コマンド、キーワードは表の切り換え
4. ユーザ名、ファイル名

内部言語方式とは、システム・メッセージをシステム・メッセージ記述言語で記述しなおし、実行時にユーザ言語に変換する方式とする。1では、システム・メッセージ記述言語で記述されたメッセージの表と変換系の追加を行った。2ではコマンド表、キーワード表の変更を行った。その他の下請ルーティンの変更は、コード変換ルーティン(小文字から大文字への変換など)、変換ルーティン(10進から2進への変換など)、パラメータ切り出しルーティン、I/Oルーティンなどに対して行った。

変換系と辞書の切り換えは、ログオン時のコマンドで選択する。

5.1 システム・メッセージ記述言語

VM/CP,VM/CMSのエラー・メッセージを解析した結果、システム・メッセージが記述すべき意味は、基本的には次の二つであることが判った。

1. 状態記述(State Description)
   システムや対象の状態に関する記述
2. 対象記述(Object Description)
   メッセージの記述対象自身に関する記述

つまり、メッセージは状態と対象の対の組合せで表現できる。システム・メッセージ記述言語の構文規則は以下である。

```
<statement> ::= <simple-statement> ;
    <statement> <conjunction>
        <simple-statement>
<conjunction> ::= AND ; OR ; THEN
<simple-statement> ::=
    <state-description> ":"
<object-description>
<state-description> ::=
    <state-predicate> ;
    <state-description> <simple-conjunction>
        <state-predicate>
<object-description> ::= <predicate> ;
    <object-description> <simple-conjunction>
        <predicate>
<simple-conjunction> ::= AND ; OR
<state-predicate> ::=
    <d/n-operator> <modal-operator> <predicate>
<d/n-operator> ::=
    <definite-operator> ; <negative-operator>
<modal-operator> ::=
    <present-tense-operator> ;
    <past-tense-operator>    ;
    <future-tense-operator>  ;
    <possibility-operator>   ;
    <necessity-operator>
<predicate> ::= <keyword> ;
    <keyword> "(" <parameters> ")"
<parameters> ::= <parameter> ;
    <parameters> ";" <parameter>
<parameter> ::=
    <keyword> ;
    <keyword> ":" <literal-list> ;
    <literal-list>
<literal-list> ::= <literal> ;
    <literal-list> <literal>
<definite-operator> ::= DEF
<negative-operator> ::= NOT
<present-tense-operator> ::= PRE
<past-tense-operator> ::= PAS
<future-tense-operator> ::= FUT
<possibility-operator> ::= POS
<necessity-operator> ::= NEC
<keyword> ::= VM ; CP ; user-id ; invalid ; ...
<literal> ::= <character string>
```

<d/n-operator> は肯定または否定を表わす。状態記述部には、時制や可能を示す項<modal-operator> が導入されている。次のメッセージは、システム・メッセージ記述言語で記述された例である。

オリジナル:
1. user-id, XXX, is not logged on.
2. user-id, XXX, is not logged on; therefore, the message is not received by XXX.

メッセージ言語:
1. NOT PRE log-on : user-id (XXX)
2. NOT PRE log-on : user-id (XXX) THEN NOT PRE receive (user-id : XXX) : message ('Are you ...')

ここでNOTは否定を表わし、PREは現在を表わす。log-on, receiveは状態を表わすキーワードで、message, user-idは対象を表わすキーワードである。

5.2 メッセージ変換系

メッセージ変換系が要求されることは、変換系自身が単純で小さく、しかもより自然なユーザ言語に変換できることである。またその際使用される辞書ができるだけ小さいことである。

内部言語で記述されたメッセージを英語や日本語に変換する方法は、基本的には次のようにする。

(日本語) <Object> が<State> である。
(英語) <Object> <State>.

ここで<Object>,<State>は対象記述、状態記述のキーワードをユーザ言語に変換した表現である。例えば前節の例を変換すると以下になる。

日本語:
1. 使用者名(XXX)が稼働していない。
2. 使用者名(XXX)が稼働していない、だからメッセージ('Are you ...')が受信されない(使用者名:XXX)。

英語:

1. User-id(XXX) is not logged-on.
2. User-id(XXX) is not logged-on; then, message('Are you ...') is not received(User-id: XXX).

メッセージ中の動詞、形容詞は状態記述として表現されるので、変換の際には注意が必要である。命令形が要求される場合には、英語の場合、

<State> <Object>

の順に変換する必要がある。例えば、

DEF NEC clear : screen

ここでNECは必要を表わす。変換されたものは、

Clear screen.
画面をクリアせよ。

以上のように、変換系は比較的簡単に実現できる。以下は、英語の変換規則である。

```
Tm(M1 AND M2) = <Tm(M1) "and" Tm(M2)>
Tm(M1 OR M2)  = <Tm(M1) "or"  Tm(M2)>
Tm(M1 THEN M2) = <Tm(M1) "therefore" Tm(M2)>
Tm(Sd ":" Od) = if imperative?(Sd)
                  then <Ts(Sd) To(Od)>
                  else <To(Od) Ts(Sd)>
Ts(S1 AND S2) = <Ts(S1) "and" Ts(S2)>
Ts(S1 OR S2)  = <Ts(S1) "or"  Ts(S2)>
Ts(d/n tense predicate) =
  <ver(d/n tense kind(predicate)
      Tp(predicate))>
To(O1 AND O2) = <To(O1) "and" To(O2)>
To(O1 OR O2)  = <To(O1) "or"  To(O2)>
To(predicate) = Tp(predicate)
Tp(keyword)   = trans(keyword, Dic)
Tp(keyword "(" parameters ")" ) =
  <trans(keyword, Dic)
      "(" trans-par(parameters) ")">
```

ここで<,>で囲まれたものは、文字列の結合を表わす。M1,M2は<statement>,<simple-statement>を表わし、Sd,S1,S2は<state-description>を、Od,O1,O2は<object-description>を表わす。imperative?は命令形かどうかを判定する関数、verは動詞の変化を司どる関数。である。trans,trans-parは辞書を引いて変換する関数。

5.3 単—2バイト文字のための変更

VM/CP,VM/CMSは370アセンブラで記述されている。システム370はバイト単位で番地付けされており、1バイト・データに対する命令が用意されている。文字の比較、移動、変換などは、1バイト・アクセスの命令を使用することが多い。従って、2バイト単位で処理するように命令を変更する。

比較的変更の多いモジュールは、

1. エラー・メッセージ・モジュール
2. 変換モジュール
3. コマンド解析モジュール

1はエラー・メッセージのテキストが、まとめられているので、ほとんど全て変更した。2はコード変換ルーティンや各種変換ルーティンをまとめたもので、文字をバイナリ・データへ、バイナリ・データを文字へ変換する下請ルーティンなどを、変更した。3はコマンド表や表探査の部分を変更した。その他のモジュールに関しては、変更が少ない。変更個所は比較的局所的であり、また文字列を扱う命令の組合せも決まったパターンが多く、ある程度自動化できると考えられる。

5.4 実験の結果

上記のメッセージ変換系をVM/CP上のエラー・メッセージに応用した。VM/CPのエラー・メッセージは1000種類以上ある。ただ、メッセージのパターンは400種類である。メッセージの生成に必要なキーワードの数は、状態記述の関するものが約200種類、対象記述の関するものが約300種類ある。

今回の実験でテストしたメッセージは、一般ユーザに対するエラー・メッセージ約100種類である。変換されたものが完全な自然言語ではないので、変換の評価を数字で表わすことはできない。メッセージの意味が理解できるという意味では十分と考える。

6 まとめ

文字セットの拡張:

日本語文字を扱う時に、2バイト文字セットが通常使用される。ヨーロッパでもヨーロッパ文字を扱うのに、2バイト文字セットを使用する例がある。台湾では、漢字が8万から10万程度使用されており、2バイト文字セット内にはおさまらず、3バイト必要と言われる。また、あらゆる文字を1つの文字セットにおさめる場合、国別コードを含めて表現する方法も提案されている。このような場合、2バイト文字セットでは不十分で、3バイトあるいは4バイト文字が必要と思われる。

拡張文字セットの対応を考えると、複数サイズの文字を混在させて扱うことは、妥当とは言えず、単一サイズの文字セットを考える方が、ソフトウェア工学的には有効と思われる。

内部言語の使用:

内部言語からユーザ言語への変換は、変換系や辞書の大きさの点で、通常のメッセージ翻訳に比べてコストが小さい。メッセージ・ライブラリと比較すると、読みやすさの点では劣るが、ライブラリ作成のコストを考えると、翻訳する必要がない分だけ有効と思われる。この方法の欠点は、ユーザにとって妥協できるメッセージが出力できるかどうか、という点である。

コマンド言語:

コマンド言語を日本語化した場合、現状では対話的に日本語を入力することは、手間のかかることであり、大抵の場合ユーザは現在のコマンド(英数字)で入力すると思われる。しかしヨーロッパ人のように、アルファベット系の文字を使用するユーザは、ヨーロッパ文字用のキーボードを用意することで、ユーロッパ言語で入力を行うことは十分考えられる。

参考文献

1. 日本語UNIXシステム諮問委員会：UNIXシステム日本語機能提案書、1985.
2. 大場、大平：抽象データ型に基づくシステム・メッセージの標準化、情報処理学会第31回全国大会.
3. 大平、川副：既存オペレーティング・システムの漢字化、情報処理学会第32回全国大会.
4. 大平、大場：多国語化を目的としたシステム・メッセージの標準化、情報処理学会第33回全国大会.
5. 川副、大平：既存エディタの漢字化、情報処理学会第33回全国大会.

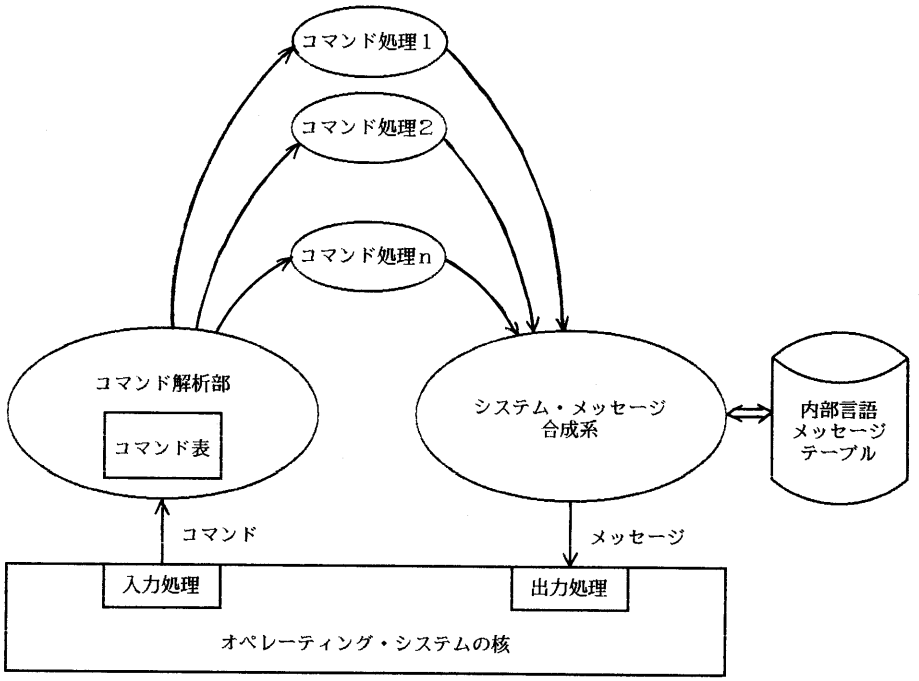

図６．実験システム