TR-I-0257

分散処理機構実現のための変換主導型機械翻訳評価実験

竹内 陽児* 古瀬 蔵

Yohji TAKEUCHI Osamu FURUSE

1992.3.31

概要

本報告では変換主導型機械翻訳 (TDMT) システムの分散並列処理の評価実験について述べる。並列処理を行なうためには各処理の実行時間を測定し、それぞれの処理を効率よく分散させる必要がある。評価実験を行なうにあたり各種のツールを作成し、3種類の並列化実験を行なった。

本報告は、学外実習生 竹内 陽児 (早稲田大学) が ATR 自動翻訳研究所 言語処理研究室で行なった実習の報告書である。

ATR Interpreting Telephony Research Laboratories

ATR 自動翻訳電話研究所

*早稲田大学大学院理工学研究科修士課程

1 はじめに

ATRでは従来より、入力文の性質に応じて必要な処理を駆動し、効率的な翻訳を実現する変換主導型機械翻訳(TDMT)システムが研究されてきた。

変換主導型翻訳[1]では、変換知識を翻訳という問題を解く鍵として位置付け、変換機構が変換知識の適用を制御する。変換機構は必要に応じて、解析知識などいろいろな種類の知識を有機的に利用して翻訳結果を作り上げる。

変換主導型翻訳の実現性や有効性を確認するためのプロトタイプシステムはGenera 8.1 Lisp Machine上で動作しており、用例距離計算[2]により高速な処理、協調的処理を実現している。しかし現在、翻訳処理は逐次処理の枠組で制御されている。従って、必要な処理を必要時に駆動するという処理を必ずしも実現しているわけではない。

今回の実験では、TDMTを共有メモリ型並列計算機のSequent上に実装し、分散処理での協調的な翻訳システムの構築のための評価実験を行なった。

まず始めに、実行時間を測定する関数を作成して逐次実行時の処理時間を測定した。次に、各種の並列化を行ない速度の向上を得ることができた。

また、並列処理の把握を用意にするためにXウィンドウシステムを用いて、実行結果をリアルタイムに表示し、実行の時間変化のグラフをポストスクリプトファイルに出力するツールを作成した。

2 TDMTの処理機構

TDMTでは、変換知識の適用が翻訳処理の中核をなし、変換知識を適用するために形態素解析、変換、解析などが協調して翻訳処理を行なう。
TDMTのシステムには現在、以下のようなモジュールがある。

- 形態素解析
  - convert-to-morph-structures 入力文を形態素へ分割し、各形態素に対し品詞・活用・シソーラスコードなどの認定を行なう。
  - lexical-transformation 意味的にまとまりのある形態素を結合するなど、形態素解析結果を翻訳処理に適するように修正する。
- 変換
  - get-english “もしもし”を“Hello.”に翻訳するように、入力文全体についてストリングレベルの変換を行なう。
  - translate-internal パターンレベルや文法レベルの変換知識を適用し、用例距離計算によって対訳、最適な構造を決定する。
- 解析
  - normalization 変換知識を適用するために、入力文を標準的な表現に正規化する。
  - local-transformation 変換知識を適用するために、特定の言語現象を入力文の局所的部分を見て検知する。
  - total-transformation 変換知識を適用するために、特定の言語現象を入力文全体を見て検知する。

逐次処理の枠組ではまず形態素解析、変換処理によって翻訳を行なおうと試み、失敗すれば解析処理を行ない、再び変換処理を行なって翻訳を行なう。次章以降では、この枠組を必要な処理を必要な時に行なう枠組に変更するための並列計算機上での基礎実験を示す。

3 実験

3.1 実行時間の測定

まず最初に,逐次処理の実行時間を測定した。ここでは、すべての処理の実行時間を測定するために、前の段階で解が求まっていても次の処理を行なうようにした。

プログラムには実行時間を記録する関数を追加し、並列化を行いやすくするために大域変数を局所変数に変更した。

実行例を以下に示す。

```
sentence : 会議で扱う話題に関して質問したいんですが
(GET-ENGLISH NIL 0)
(CONVERT-TO-MORPH-STRUCTURES 650)
(LEXICAL-TRANSFORMATION 133)
(TRANSLATE-INTERNAL 2300)
(TRANSFER 2317)
  NIL
(NORMALIZATION 50)
(LOCAL-TRANSFORMATION 150)
(TOTAL-TRANSFORMATION 50)
(TRANSLATE-INTERNAL 4933)
(ANALYSIS&TRANSFER 5200)
  ( I WOULD LIKE TO ask ABOUT topic treated AT conference .
 0.7708334)
  ( I WOULD LIKE TO ask ABOUT topic treated AT conference .
 1.2187501)
  ( I WOULD LIKE TO ask ABOUT topic treated AT conference .
 1.2708334)
  ( I WOULD LIKE TO ask ABOUT topic treated AT conference .
 1.2708334)
  ( I WOULD LIKE TO ask ABOUT topic treated AT conference .
 1.7187501)
(TRANSLATE 8400)
```

また、実行の様子を図 1に示す。

図 1: 実行時間の測定 (例)

3.2 入力文の分類

実行時間の測定の実験から入力文の分類を行なった。実際の処理では必要な処理だけ実行すれば良いので、入力文がどこまでの処理を必要としているかによって分類することは、並列化を行なう上で重要な情報となる。

分類を以下に示す。

1.	全ての文	225
2.	重複を除いたもの	165
3.	2. のうち、get-english で答の求まらないもの	148
4.	3. のうち、analysis の前と後の結果が同じもの	100
5.	3. のうち、transfer で答の求まらないもの	48
6.	3. のうち、transfer と analysis&transfer の結果の異なるもの	6

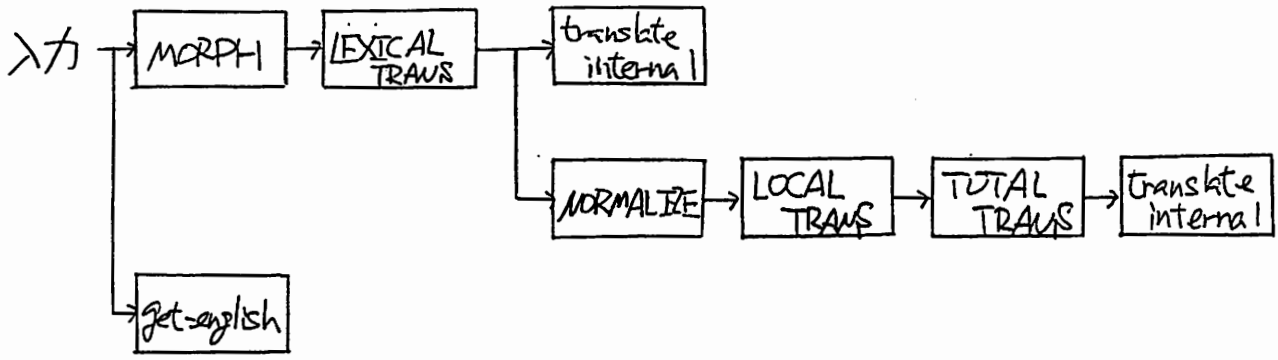

図 2: データ依存グラフ

3.3 並列化その 1

データ依存グラフを図 2に示す。データの依存関係から get–english と convert–to–morph–structures→ lexical-transformation→ $\cdots$ を並列に実行し、 translate–internal と normalization → local–transformation → $\cdots$ → translate–internal を並列に実行できることがわかるが、 get–english は, 実行時間がほぼ 0msec であるため、並列に実行しても意味がない。また、並列処理によるオーバヘッドによって, かえって実行時間が大きくなってしまう。

したがって、まず get–english を行ない、その結果によって後の処理を行なうか決定する。実行する場合は、 convert–to–morph–structures を実行した後、 translate–internal と normalization → local–transformation → $\cdots$ → translate–internal を並列実行する。

実行例を以下に示す。

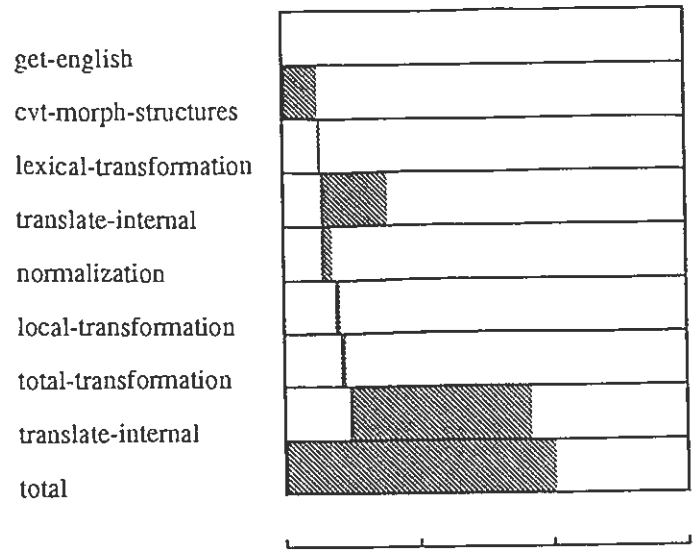

図 3: 並列実行その 1(例)

```
sentence : 登録用紙を至急送らせて頂きます
(GET-ENGLISH NIL 0)
(CONVERT-TO-MORPH-STRUCTURES 284)
(LEXICAL-TRANSFORMATION 17)
(TRANSLATE-INTERNAL 534)
(TRANSFER 550)
  ( WE WILL send IMMEDIATELY registration form . 5.0e-31)
(NORMALIZATION 100)
(LOCAL-TRANSFORMATION 67)
(TOTAL-TRANSFORMATION 34)
(TRANSLATE-INTERNAL 1334)
(ANALYSIS&TRANSFER 1583)
  ( WE WILL send soon registration form . 0.625)
  ( WE WILL send soon registration form . 1.125)
(TRANSLATE 2020)
```

また、実行の様子を図 3に示す。

以後は, translate-internal を transfer、 normalization → local-transformation → total-transformation を analysis、 normalization → local-transformation → ··· → translate-internal を analysis&transfer とおくことにする。

今回の処理では、transfer と analysis&transfer を同時に実行し両方の結果を受けとることにした。こうすることにより、transfer で答が出ない場合には、同時に analysis&transfer を実行しているためのその分早く答を得ることができるようになった。しかし、analysis 前と後の結果が同じ場合 analysis&transfer の translate-internal が analysis の分だけ遅れて実行されるので、同じ結果を得るのを待つために実行時間がかかってしまう。

3.4 並列化その2

ここでは、前の実験での欠点の改善について述べる。前の実験での問題点は、transfer と analysis&transfer の両方の答が返ってくるまで処理が終えられないということであった。そこで、答が1つ返ってきたら処理を終了するための変更を行なった。

最初は、transfer と analysis&transfer を同時に実行し、答が得られたら実行中のプロセスを終了させる方法をとっていた。しかし、この方法では頻繁にデッドロックが発生してしまうため、断念した。これは、Sequent Symmetry Allegro CLiP の相互排他の機構が不完全であるためと思われる。

次に用いた方法は、答が得られたら実行中のプロセスはそのままで、メインプロセスは答を出力して先に終了してしまうやり方である。この方法では、プロセスが1つ残されたままになるが、答を得るまでの時間は最初のものと同じにすることができる。

実行例を以下に示す。

```
sentence : 登録用紙を至急送らせて頂きます
(GET-ENGLISH NIL 0)
(CONVERT-TO-MORPH-STRUCTURES 283)
(LEXICAL-TRANSFORMATION 33)
(TRANSLATE-INTERNAL 533)
(TRANSFER 550)
  ( WE WILL send IMMEDIATELY registration form . 5.0e-31)
(NORMALIZATION 100)
(LOCAL-TRANSFORMATION 50)
(TOTAL-TRANSFORMATION 50)
(TRANSLATE-INTERNAL 1283)
(ANALYSIS&TRANSFER 1500)
  ( WE WILL send soon registration form . 0.625)
  ( WE WILL send soon registration form . 1.125)
(TRANSLATE 920)
```

また、実行の様子を図4に示す。

しかし、これらの方法でも問題が生じることになった。transfer と analysis&transfer の実行結果が異なる分類6の入力文の場合、片方の答しか得ることができない。また、2つの処理の実行時間が似通っている場合、どちらの答が先に得られるかは実

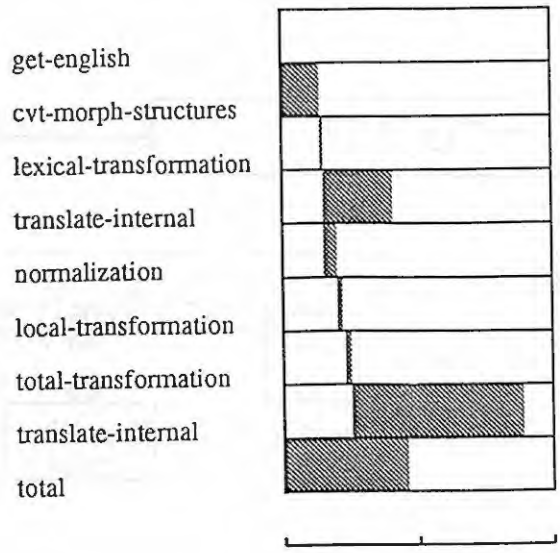

図 4: 並列実行その 2(例)

行するたびに変わってしまうのである。

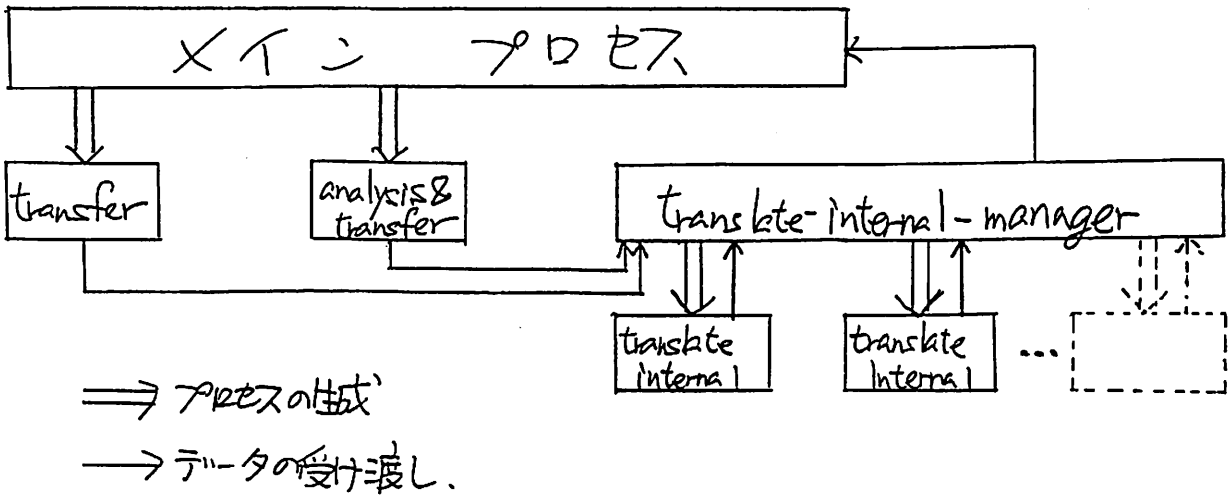

図 5: 動的なプロセスの生成を導入した並列化の過程

3.5 並列化その 3

動的にプロセスを生成するという汎用的な枠組を導入する。

transfer と analysis&transfer で行なわれている translate-internal を統括するプロセスを作る。この新しいプロセスは自分宛の mail-box を監視して、transfer や analysis&transfer から translate-internal の実行の要求があった場合は過去に同じ処理の要求があるかを調べ、ない場合には新たにプロセス transfer-internal を生成する。そして、いくつかの生成した transfer-internal からの処理結果を受けとり、結果を統合してメインプロセスに送る。その様子を図 5に示す。
実行例を以下に示す。

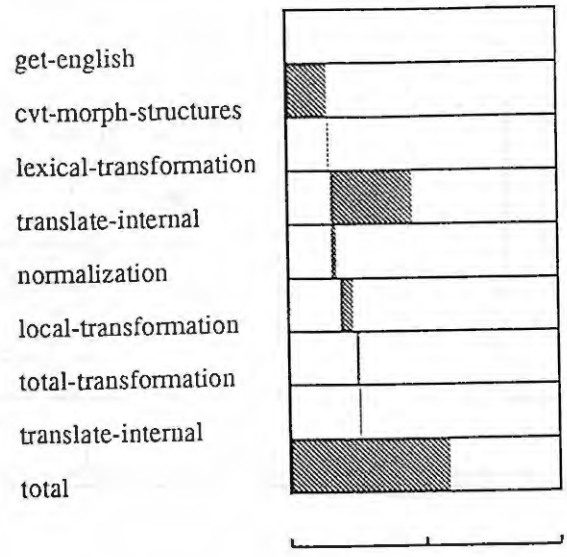

図 6: 並列実行その 3(例)

```
sentence : 会議について詳しいことを教えて下さい
(GET-ENGLISH NIL 0)
(CONVERT-TO-MORPH-STRUCTURES 316)
(LEXICAL-TRANSFORMATION 17)
(TRANSLATE-INTERNAL 617)
  ( PLEASE tell the details ABOUT conference . 5.0e-31)
  ( PLEASE tell the details ABOUT conference . 0.5)
(NORMALIZATION 50)
(LOCAL-TRANSFORMATION 117)
(TOTAL-TRANSFORMATION 33)
(TRANSLATE 1190)
```

また、実行の様子を図 6に示す。

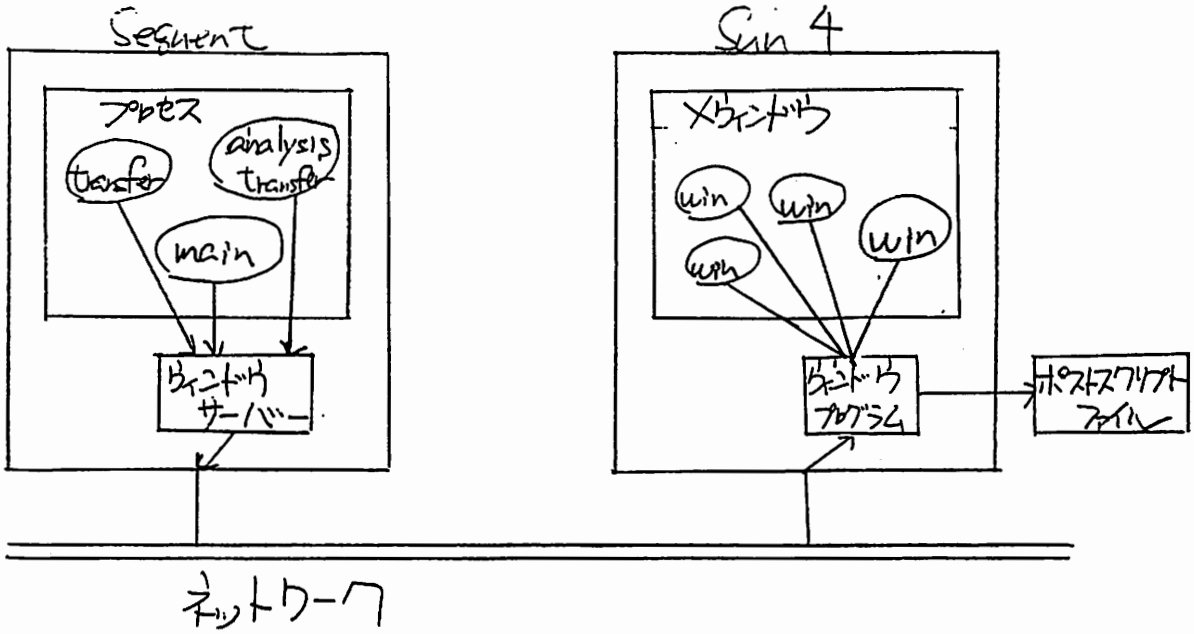

図 7: ウィンドウシステムの構成

4 ウィンドウシステム

今回の実験では並列処理の実行の把握を容易にするためのユーザーインターフェイスにXウィンドウシステムを用いた。Sequent Symmetry S81 と Sun4/260 はイーサネットを通じてソケット通信を行ない、処理結果はSun4のXウィンドウ上にリアルタイムに表示される。また、実行の時間変化はウィンドウプログラムからポストスクリプトのファイルとして出力される。(図 7、8参照)

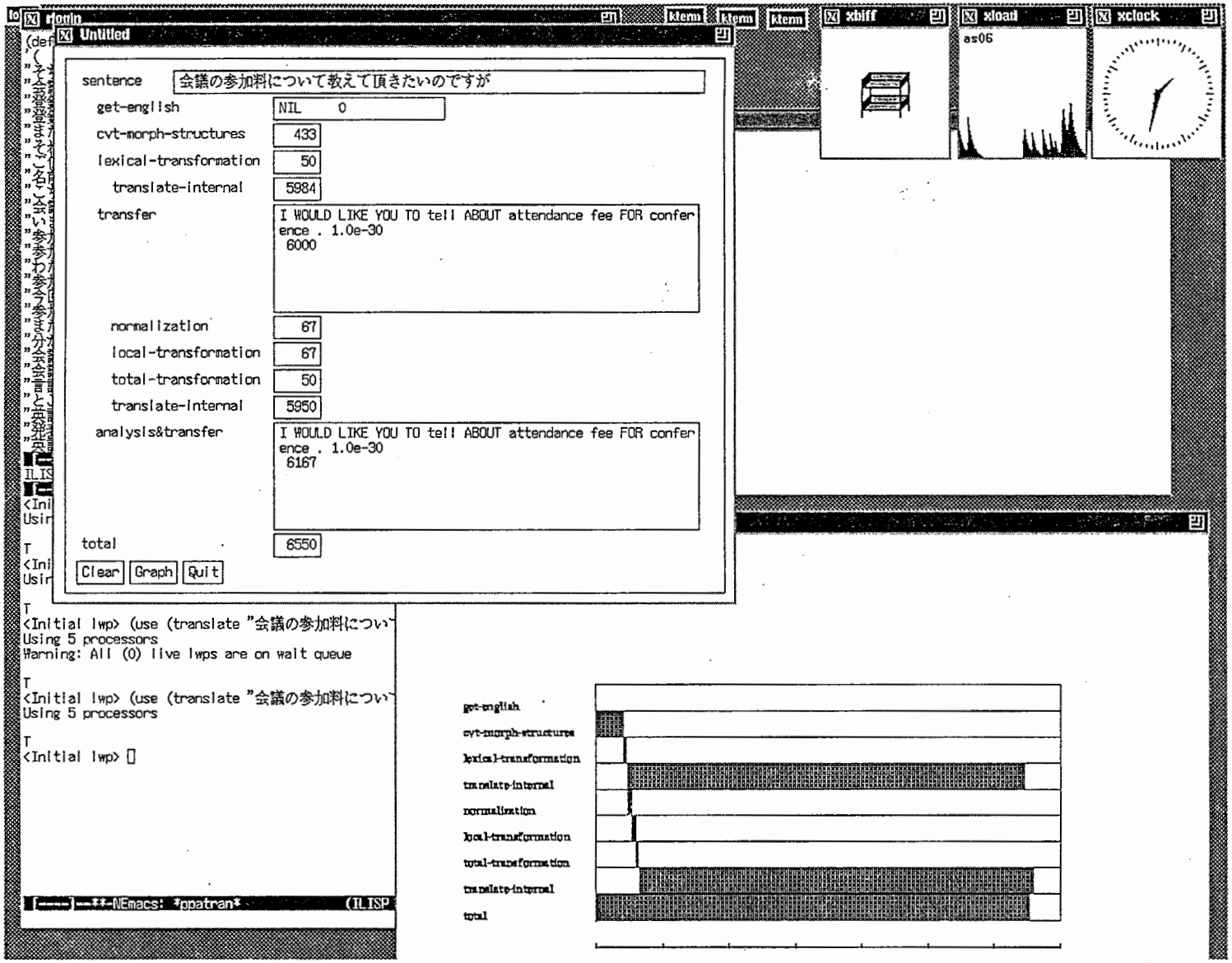

図 8: ウィンドウシステムの例

5 まとめ

今回の実験の結果、変換処理の過程において必要な段階でプロセスを動的に生成するという枠組の有効性が確認された。今後はより高速化をはかるために1つのプロセスとして実行していたtranslate-internalをさらに並列化する必要があると思われる。

また、より多くのプロセスが協調的に情報を受渡しするようになると、共有メモリに対するアクセスのオーバヘッドが大きくなると思われ、より効率的な情報の交換機構が必要になると思われる。

6 謝辞

本研究の機会を与えて頂いた言語処理研究室室長の飯田 仁氏に感謝致します。また、熱心に御指導して頂いた言語処理研究室の皆様に感謝致します。特に、Lisp言語初心者の私の質問に親切に答えて頂くなど、数多くの助言をして頂いた、大熊 英男氏に感謝致します。

参考文献

[1] 古瀬, 飯田：変換と解析の協調処理による翻訳手法, 情報処理学会自然言語処理研究会報告,87-4,(1992).

[2] Sumita, E., and Iida, H : Experiments and Prospects of Example-based Machine Translation, Proc. of the 29th Annual Meeting of the Association for Computational Linguistics,(1991).