Chapter 8

配送計画支援システム Navimos の開発

ITS(Intelligent Traffic System) [87] は、官民が協力して推進している交通環境改善のための事業である。その中でも、総交通量の大半をしめる物流交通を情報技術によって最適化しようとする試みは、ITS の物流交通への適用と位置付けられている。特に、物流コストの低減は、単なる物流業者のコスト意識だけでなく、環境保護活動の面からも早急な対応を迫られる課題となっており、情報技術の活用による物流コスト低減の効果が期待されている。

本章では、オブジェクト指向を用いた物流交通の最適化を支援するアプリケーションフレームワークの構想と、その部分的な解決事例、および Nuts コンポーネントを用いたアプリケーションフレームワークの構築について述べる。

8.1 物流交通アプリケーションフレームワークの構想

物流分野では、取り扱う荷、荷姿、輸送距離、輸送形態、輸送機種、会社規模、などの違いによって、様々な業務形態が存在しており、定型的な情報システムではカバーしきれていない。そのため各物流業者は、業務毎に個別システムを特注するか、物流パッケージのソフトウェアを大幅にカスタマイズすることにより対応している。しかし、このような個別の対応では最適なシステムを迅速に手に入れるのは困難である。

そこで、Nuts を用いて汎用的な物流ドメインのためのアプリケーションフレームワークを構築し、その上で、様々な業務形態に特化したシステムを迅速に開発できるようにすれば、情報システムの活用が物流コスト低減に効果を発揮すると考えられる。本節では、物流ドメインのアプリケーションフレームワーク Navimos [73] [74] の構想を説明する。著者はまず、既存のフレームワークを用いて Navimos のサブシステム Navimos-Master(Win32/MFC 版) を先行開発した。その後、物流ドメインのアプリケーションフレームワークとしての柔軟性を高めるため、Nuts を用いた Navimos-Master(Nuts 版) を開発した。本節で、Navimos の基本構想を説明した後、第 8.3節で Win32/MFC 版 Navimos の開発について、第 8.4節で Nuts 版 Navimos の開発についてそれぞれ説明する。両者の開発効率の差異については、第 9.1.2節で議論する。

Navimos フレームワークでは、物流分野における以下のような問題をサポートする。

1. 配送最適化支援—現在主流であるトラック物流において、配車および配送コースを最適化することにより、トラック台数の削減、走行距離/稼働時間の削減を目的とした最適化を行う。

2. 配置最適化支援—物流拠点 (ロジスティクスセンター)、倉庫などの配置を最適化し、輸送効率の最適化を図る。

3. 動態管理—トラックなどの移動体に対して、交通情報を反映させた配送コース最適化や位置管理を行うことにより、動的に物流交通の最適化を行う。

4. 日常業務支援—配車指示、日報記録など日々の業務を支援する。

Navimos フレームワークの目的は、上記のような問題をサポートする各種アプリケーションを柔軟かつタイムリーに開発するために、物流分野で共通に用いる、配送先、工場、倉庫、トラックなどの現実の物、及び受発注、出庫、配送指示などの手続き、地図、タイムテーブル、伝票などの帳票類を抽象的に定義し、様々な場面で再利用可能なアプリケーションフレームワークを構築することである。

8.2 フレームワーク Navimos とは

Navimos は、物流分野における様々な支援アプリケーション群を開発するための、アプリケーションフレームワークである。Navimos では、物流分野で共通に用いられる要素とそれらの関連を抽象的に定義し、様々な物流形態に柔軟に対処可能なフレームワークとする。

Navimos のアプリケーションは図 8.1のような 3 つの分野をサポートする。

Navimos-Master 配車/配送コースの最適化ツール。配送のための様々な条件を満たした上で最適な配車計画を立案する。主に運送業者をサポートする。

Navimos-Lite 物流のための車載ナビゲーションシステム。電子的に受け取った配送計画をもとに、交通情報、移動履歴などを考慮して車上での配送計画最適化を行う。ドライバーをサポートする。

Navimos-Link Navimos-Master の静的な配車計画、Navimos-Lite の動的な配送計画の情報をインターネットを通じて荷主や輸送基地に配信するシステム。不特定多数のアクセスポイントから、荷やトラックの情報が入手でき、空荷の防止などの最適化に役立つ。主に荷主をサポートする。

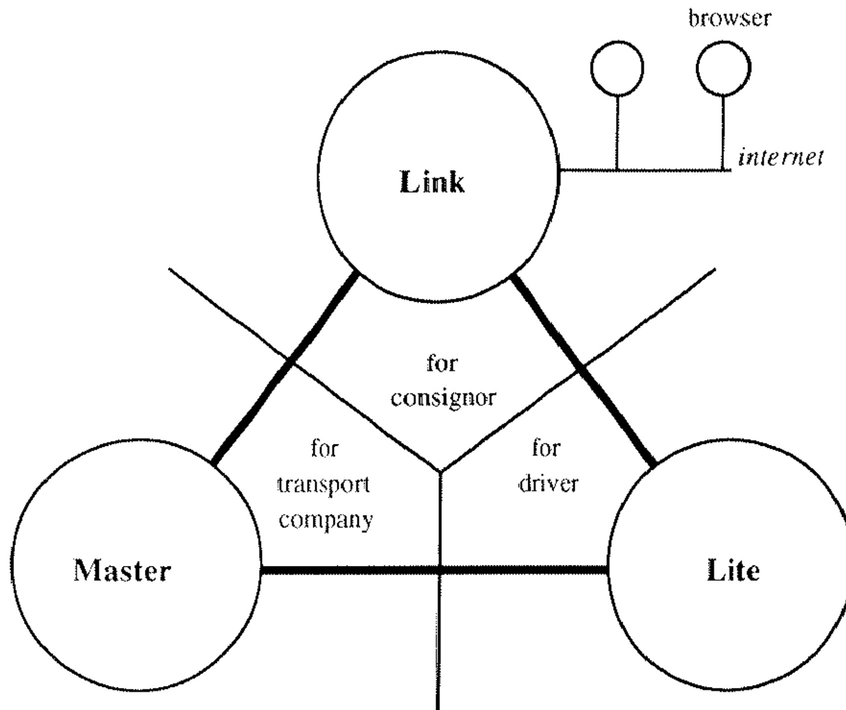

図 8.1: Navimos のサポート分野

8.3 配送計画支援システム Navimos-Master の開発

著者はまず、配車／配送コースの最適化ツールである Navimos-Master に対して、巡回配送の最適化問題に絞ったシステムの開発を行った。

今回のシステムでは、日用品等を小売店に巡回配送するような小口の運送を対象とする。このような配送は、小型トラックの得意とするところであるが、そのようなトラックを多数抱える配送センターでは、日々変動する荷量や制約条件の組合せが複雑となり、最適な配送計画の集中管理が困難となっている。そこで、配送計画の作成を計算機シミュレーションを用いて支援し、配車担当者の経験を活かしながら対話的に配送計画の立案が可能なシステムの開発を行なった。

8.3.1 最適な配送計画とは

配送センターに集約された荷を、主に 2t 車クラスの小型トラックで、一台あたり平均 20〜30 の配送先を巡回して配送する場合を考える。これをトラック台数 30 台を有する配送センターで見た場合、総配送先数は 600 箇所/日を越える大規模なものとなる。

配送計画は、

- どのトラックがどこに配送するかを割り当てる— 配車計画

- それぞれのトラックがどのような順序で配送するかを決定する— 配達計画

に分けられる。配送計画の制約条件を整理すると以下のようになる。

- トラックの総積載量 (体積, 重量) を越えない。
- トラックの稼働時間の上限を越えない。
- トラックの配送先数の上限を越えない。
- 決められた時間の休み時間を取らなければならない。
- 各配送先には配送指定時間があり厳守しなければならない。

以上の制約条件を満たした上で、各トラックの稼働時間、走行距離が最小となるような配車/配達計画を最適な配送計画と呼んでいる。

前述したように、配送センターでは 30 台以上のトラックを持つ。これらのトラックが、600 箇所以上の配送先の属性 (場所, 荷下ろしにかかる時間, その日の荷量など) を考慮した上で最適な配送を行えるような計画を人手によって立案するのは困難である。その結果、配車計画は制約条件を破らない程度に余裕を持たせたものにならざるをえず、積載率の恒常的な低下、トラックの余剰所有、ドライバーの超過勤務、などが効率化の妨げとなっている。また、配達計画はドライバー任せになる場合が多く、センターで配達ルートを把握できないなどの弊害を生んでいる。さらに、優れた配送計画担当者の育成、確保が必要であり、担当者への依存が配送計画の質の平順化を難しくしている。そこで、本システムでは、計算機を用いて組合せ最適化問題を解くことにより最適な配送計画の立案を支援する。

8.3.2 最適化のアルゴリズム

8.3.2.1 配車最適化

Navimos-Master の配車計画は、セービング法をベースとしている。本節では、まずセービング法について説明する。

配送センターを $node_0$、$node_i$から $node_j$へ行くためのコスト (距離など) を C_{ij}とした場合、$node_i$と $node_j$のセービング値を

$$S_{ij} = C_{0i} + C_{0j} - C_{ij}$$

と定義する。これは図 8.2に示すように、センターから $node_i$と $node_j$をそれぞれ往復した場合と、センター→ $node_i$→ $node_j$→センターと巡回した場合のコストの短縮幅を表している。セービング法では、すべてのノードの組合せに対してセービング値を計算し、セービング値の大きい順にノードを合併していく (図 8.3a)。合併する際には、そのノードを合併することによって一台のトラックの荷量や最大配送先数などの上限を越えないか検査する。

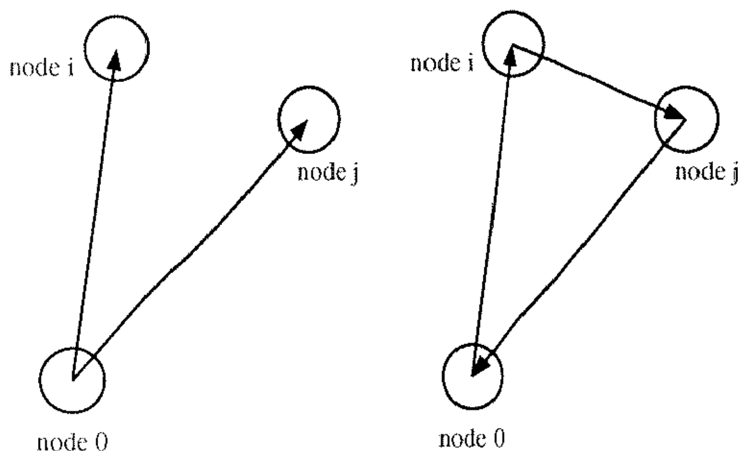

図 8.2: セービング値

検査に合格しない場合は合併しない。合併の結果できたノードのグループに一台のトラックを割り当てると配車が完了する。

しかし、単純なセービング法では各ノードに時間制約がある場合、正しい配車ができない。なぜならば時間制約が守れるかどうかは、配送順序 (配達計画) が確定してからでなければ判別できないためである。例えば、同じ時間制約に属するノードが一台のトラックに割り当てられるというような矛盾した配車になる場合がある。

そこで、合併判断において、あるノードを加えた場合の仮の最適な配送順序を決定し (後述する 2-opt メタヒューリスティックス解法による)、時間制約を破る場合は合併しないという戦略を採用した (図 8.3b)。これにより時間制約を満たした配車計画が可能になった。

8.3.2.2 配送コース最適化

Navimos-Master の配達計画 (配送順序の決定) は、配車計画で求まった配送先を最小のコストで巡回するためのルートを決定する。配達計画には、巡回セールスマン問題 (TSP) の解法の一つである、2-opt メタヒューリスティックス解法を用る。これは図 8.4のように、配送コース上の 2 つのリンク i,j を交換した場合の目的関数の変化量 D_{ij} を計算し、最も変化量の大きいリンクを交換するという手法である。

具体的に以下のようなステップを踏む。

1. 時間制約が守られず遅れが生じている場合は、遅れが最も小さくなるリンクを交換する。

2. 時間制約が守られている場合は、時間制約を守るものの中で最も帰着時刻が早くなるリンクを交換する。

3. 目的関数 (時間遅れか稼働時間) の変化量 D_{ij} が 0 になるか、決められた反復回数に到達までリンクの交換を繰り返す。

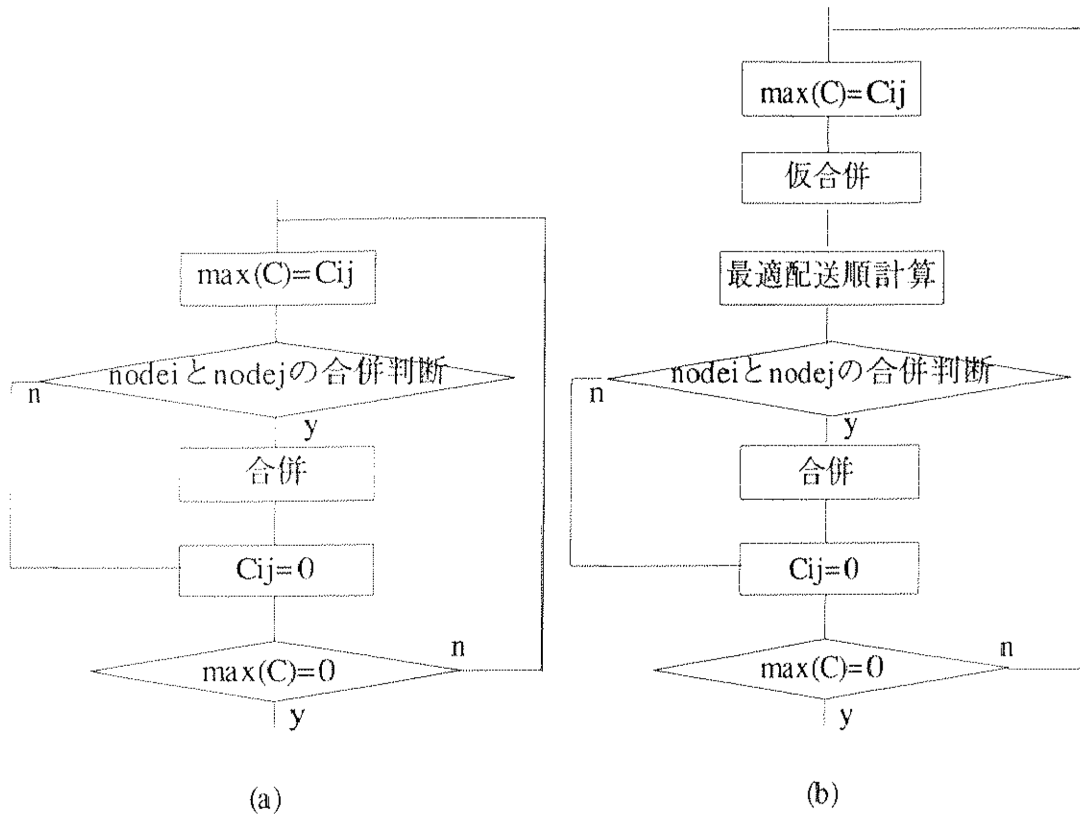

図 8.3: Navimos の時間制約付きセービング法

8.3.3 データの準備

配送先の属性として、配送先ID、住所、電話番号、位置 (経度, 緯度) を 1 レコードとしたデータベースを用意する。この中で配送先の位置は特に重要である。Navimos-Master では、トラックの稼働時間を目的関数に使用しているため配送先間の所要時間を知ることが必要になる。現在の仕様では、$node_i$と $node_j$ 間の経度, 緯度から求めた直線距離を d_{ij}、平均時速を v_{ij} とした場合、所要時間 t_{ij}を

$$t_{ij} = \frac{d_{ij} \times 1.25}{v_{ij}}$$

で求めている。係数 1.25 は、直線距離と道のり距離の関係を実測サンプルを用いて重回帰分析により求めた結果である。

平均時速は、センターとの間、およびそれ以外、で別々に設定できるようになっている。これは一般に、センターから配送エリアまでは高速道路などを使って高速に移動できるが、エリア内では小刻みに停車／発進を繰り返すため車速が落ちることを反映するためである。

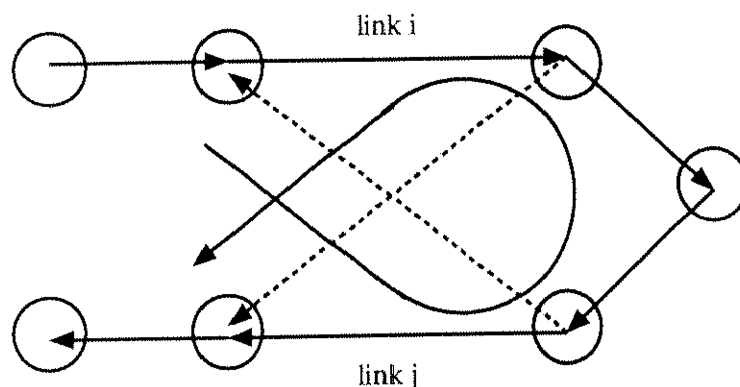

図 8.4: 巡回セールスマン問題のメタ解法

例えば、典型的な値として、

$$v_{ij} = \begin{cases} 20km/h & i = 0 \\ 8km/h & \text{otherwise} \end{cases}$$

を使用できる。これらの値は経験に基づいたものとなっている。

8.3.4 計画提示と修正

Navimos-Master の操作パネルを図 8.5に示す。Navimos-Master の配車、配達計画は以下の 3 つのビューを用いて視覚的に表示される。

- 地図表示—地図上で配送ルートを直観的に把握することができる。配送順序を直線で結ぶ他に、道路マッチング機能によって実際のルートに当てはめて見ることもできる。
- タイムテーブル表示—ガントチャート (帯グラフ) によって各ノードの到着時間/出発時間、荷役時間、荷量を同時に把握することができる。
- リスト表示—各ノード、ルートの情報を詳細に表示する。

これらの 3 つのビューは連動しており、例えばタイムテーブル上のノードを選択すると、そのノードの地図上、リスト上での位置がマークされるので、関連する情報を簡単に取得できる。

さらに Navimos-Master では、配送計画を簡単な方法で修正することができる。配車を修正したい場合、タイムテーブル上でノードをドラグして他のトラックのルートにドロップすることで可能になる。配送順序を修正したい場合は、同じくタイムテーブル上でノードをドラグし他のノードの上にドロップすると、そのノードの次に配送されるようになる。

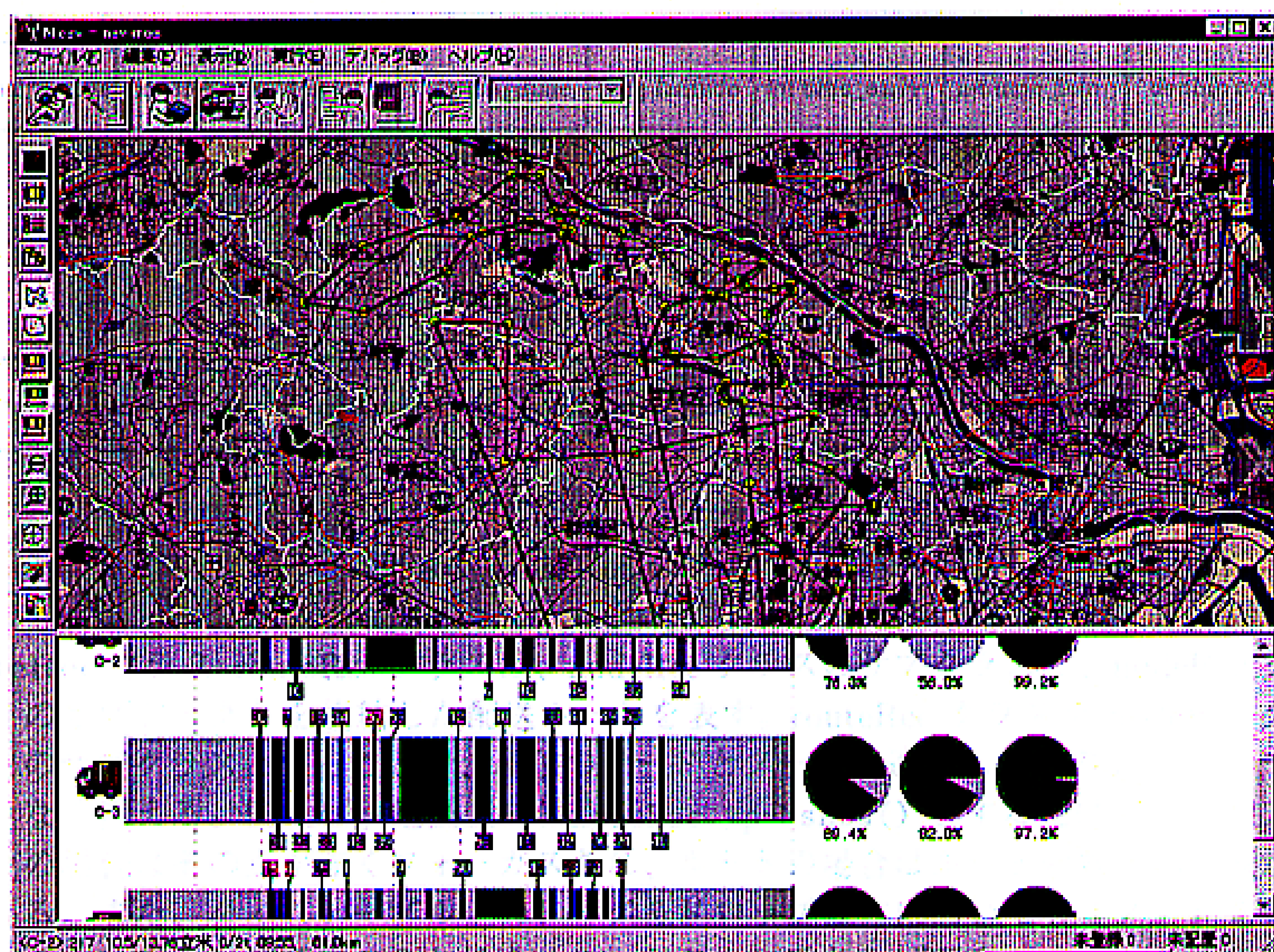

図 8.5: Navimos-Master の操作パネル

このように、Navimos-Master では計算機アルゴリズム万能の立場ではなく、最終的に配送計画担当者の経験や都合を採り入れることができるインタフェースを用意することで、柔軟な配送計画が可能なシステムとなっている。

8.3.5 Navimos-Master の実装

Navimos-Master では、配送先, トラックなどを扱うデータ (data クラス)、データを用いてシミュレーションを実行するアルゴリズム (algorithm クラス)、シミュレーション結果を表示するビュー (view クラス) をそれぞれ抽象的に定義し、実務形態に合わせてサブクラス拡張を行う。これにより Navimos-Master は、さまざまは表示形態、データ表現に対応し、またアルゴリズムの変更も容易になる。以下に Navimos-Master の実装で定義した主なクラスについて説明する。

nodeRec クラス nodeRec クラスは、data クラスのサブクラスであり、配送先や配送基地など地図上の点を表す。図 8.6のデータメンバーを持つ。自分が属しているルートへの参照を持っている。

char *name	nodeREc *route	int id	long longitude	long latitude	int time_early	int time_late	double weight

図 8.6: nodeRec クラスのメンバー

char *name	int id	double weight	uint property	int time_start	int time_rest	int rest_early	int rest_late	routeRec *route

図 8.7: truckRec クラスのメンバー

routeRec クラス routeRec クラスは、nodeRec クラスのサブクラスで、nodeRec クラスのオブジェクトを連結した配送ルートを表す。routeRec クラスを nodeRec クラスのサブクラスとすることで、単独のノードとノードを連結したもの (ルート) を区別なく扱える。例えば、地図上では`nodeRec *p->show()` とするだけで、単独ノードの場合はその点が赤くフォーカスされ、ルートの場合はルートの線が赤く表示される。また、タイムテーブル上のドラグ&ドロップ操作においても「ノードをノードの上に重ねれば順序の入れ換え、ルートの上に重ねれば配車の変更」という動作を `nodeRec *p->drop(nodeRec *q)` という統一的な方法で行える。

truckRec クラス truckRec クラスは data クラスのサブクラスで、図 8.7に示すようなトラックの属性を保持する。これらの属性は truckRec クラスに附属するダイアログ (図 8.8) によって実行時に適宜変更可能なようになっている。トラックが、どのルートに割り付けられたかを示すために routeRec への参照を保持している。

mapView クラス mapView クラスは、view クラスのサブクラスで地図データベースを参照してシミュレーション結果を電子地図の上に視覚的に表示する。地図上に点を打つ `addSpot()`、縮尺を返す`changeScale()` などのメソッドがある。実際には、使う地図データベースに合わせて、地図データベースへのアクセス関数を呼び出すようなサブクラスを定義して用いる。mapView クラスを利用することにより、様々な地図データベースの利用や、地図データベース自体の更新に対処可能である。

timeTableView クラス timeTableView クラスは、view クラスのサブクラスで、巡回配送のスケジュールをタイムテーブルで表示する。トラックの稼働時間、荷役時間、荷量などが同時に把握できる。また、ドラグ&ドロップ操作で配車や配送コースの変更が容易に行える。

3DtimeTableView クラス 3DTimeTableView クラスは、timeTableView クラスのサブクラスである。timeTableView クラスでは、ガントチャートの横軸を時間、縦軸を荷量

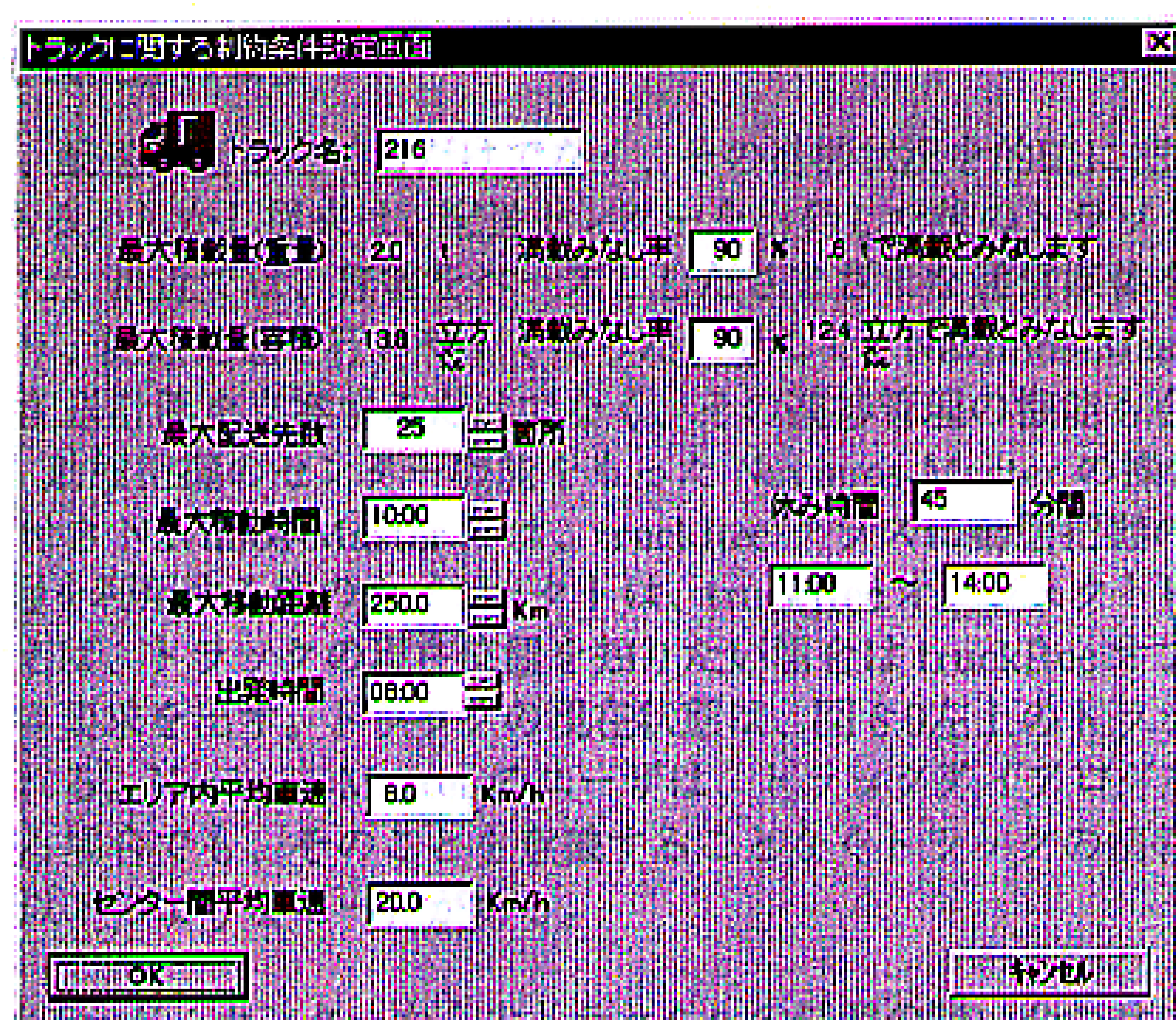

図 8.8: トラック属性の変更ダイアログ

として表現しているが、3DTimeTableView では、荷量を Z 軸方向に 3 次元的に表示する。その他は、timeTableView の機能を継承する。

SavingMethod クラス SavingMethod クラスは algorithm クラスのサブクラスで、セービング法により配車計画を立案する。algorithm クラスのメソッド `execute()` をオーバーライドしており、セービング法の実行を開始する。セービング法の途中で、以下の optMethod クラスのオブジェクトにより TSP(Traveller Sailsman Problem:巡回最適化) を解決する。

optMethod クラス optMethod クラスは algorithm クラスのサブクラスで、2-opt メタヒューリスティクス解法により TSP を解く。algorithm クラスのメソッド `execute()` をオーバーライドしており、TSP の実行を開始する。

今回開発した、Navimos-Master は Microsoft Window95/98/NT 上で動作する。物流現場への低コストでの導入を考慮して、現在最も普及している Window95/98/NT PC を実行環境に選択した。開発には Microsofot Visual C++の統合開発環境上で、MFC ライブラリを用いた。コード量はおよそ 19,000 行である。

8.3.6 Navimos-Master への動態把握機能の追加

8.3.6.1 動態把握機能の概要

物流業務では、配送計画をたてるだけでなく、その進捗状況を的確に把握すること (動態把握) が重要である。しかし従来は、配送センターを出発した後のトラックの足取りを掴むのは困難であった。最近では、無線をベースとしたさまざまな通信媒体を手軽に利用することが可能となり、動態把握を行いやすくなった。そこで、Navimos-Master に動態把握の機能を追加し、配送計画を立案するだけでなく、その進捗管理の機能も合わせ持つようにした。図 8.9が、動態把握のセンター機能をもった Navimos-Master の操作画面である。

動態把握機能付き Navimos-Master では、truckRec クラスに動態把握の機能をカプセル化する。動態把握を利用する他のオブジェクトは、truckRec オブジェクトに対して情報取得要求を出す。例えば、トラックの現在位置を知りたい場合は truckRec オブジェクトに対して、現在位置取得要求を出す。トラックの現在状態は、図 8.9左上のトラックダイアログに集中して表示される。Navimos-Master のユーザは、トラックダイアログを使って、トラックが遠隔地にいる場合でも、そのことを意識することなくトラックの情報を取得できる。

動態把握には以下のようなモードがある。

- 問い合わせモード：センターからのトラックへの任意の問い合わせにより情報を取得するモード。以下のような情報が取得できる。
  - 現在位置 (緯度／経度, 高度)
  - GPS ロケータの状態 (電界強度, 時刻, 捕捉衛星数など)
  - 平均車速／走行距離
  - 何番目の配送先まで配送したか
  - 現在の遅れ／進み時間
- ポーリングモード：トラックが一定の条件を満たした時に、自動的にセンターに通知するモード。以下のような条件がある。
  - 配送先に到着した時
  - 配送先を出発した時
  - 一定時間以上の遅れが発生した時
  - 一定時間以上平均車速が規定値を下回った (渋滞した) 時

8.3.6.2 動態把握機能の実装

Navimos-Master と動態 (トラック) との間は無線によるデータ交換を行う。現在、無線による通信手段にはさまざまものが考案／実用化されており、それぞれ利点／欠点を持っている。Navimos-Master は、特定の通信手段に依存せず、また将来の新たな通信手段も容

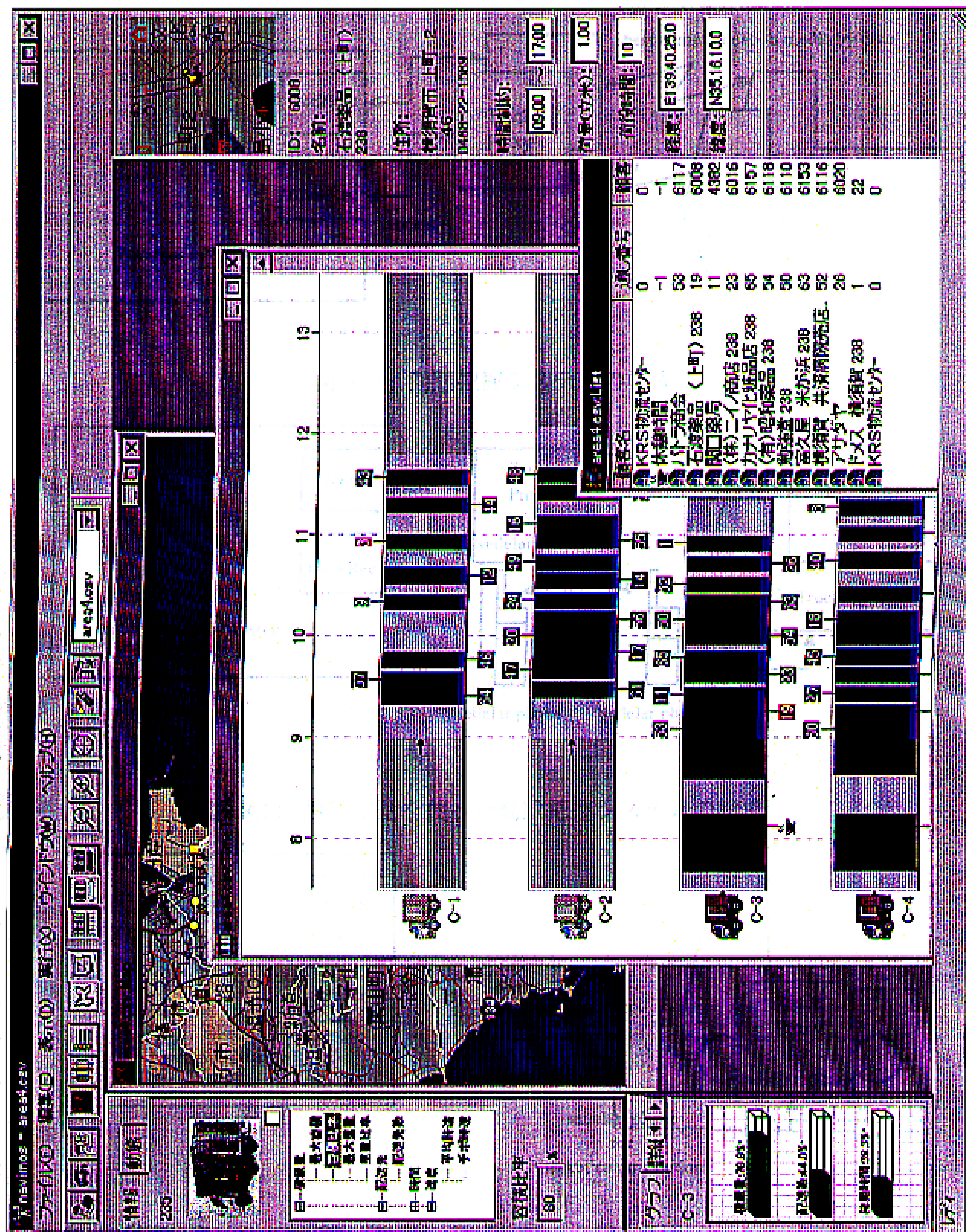

図 8.9: 動態把握機能をもった Navimos-Master

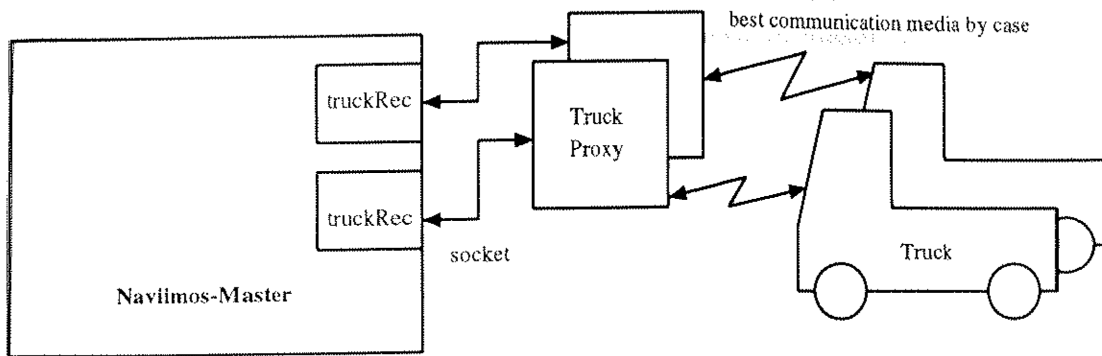

図 8.10: 動態把握システムの構成

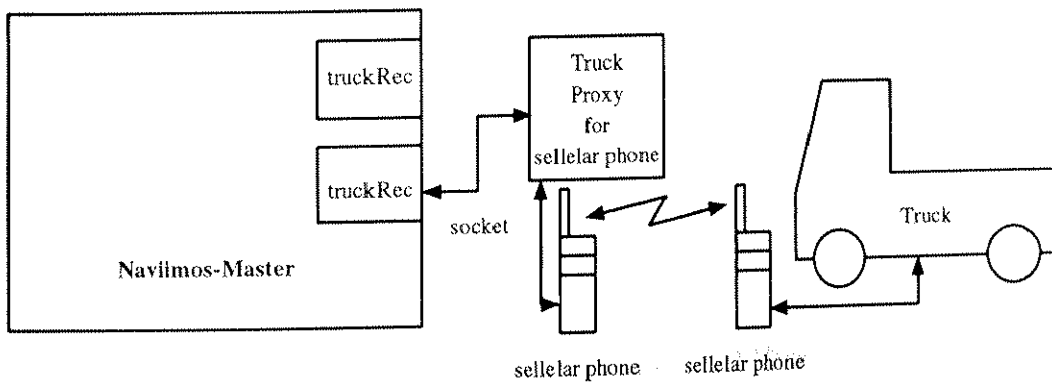

図 8.11: 携帯電話を使った動態把握システムの構成

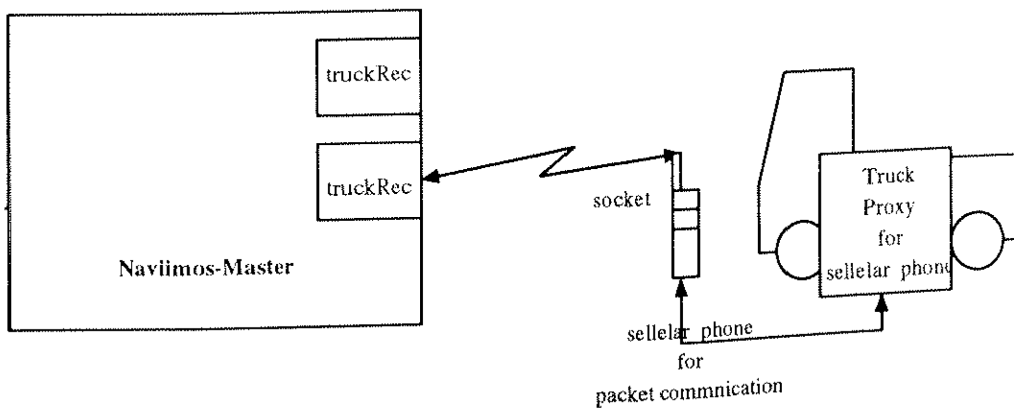

図 8.12: パケット通信を使った動態把握システムの構成

易に取り込めるように、truckRec クラスの中で抽象化した通信機能を用いている。また、通信処理には話中や接続の途中切断など不安定な要素があり、通信処理を Navimos-Master に直接実装するのは得策でないと判断した。そこで、Navimos-Master では、トラックを表す代理人 (Proxy) を導入した。

トラック Proxy は、Navimos-Master とは別のプロセスであり、IP アドレスを持った別のマシン上 (同じマシン上でもよい) に存在する。Navimos-Master は、通常のプロセス間通信 (Socket を利用) によって、トラック Proxy とデータのやりとりをする (図 8.10)。Navimos-Master は、トラック Proxy が実際にどのマシン上にあるのか、あるいはどのような通信手段を用いているのかを意識する必要がない。

例えば、通信手段に携帯電話のデータ通信を用いるのであれば、図 8.11のような構成になる。トラック Proxy は、Navimos-Master からの要求に基づいて、トラック上の携帯電話と AT コマンドを使ってデータをやりとりするが、Navimos-Master から見た場合は、単なる Socket でアクセス可能なプロセスにしか見えない。つまり、この場合のトラック Proxy は、携帯電話によるデータ通信をカプセル化している。

また、NTT-DoCoMo が開発したパケット通信網 DoPa を利用する場合は、図 8.12のように、トラック Proxy がトラック上にそのまま搭載された構成になる。Navimos-Master からは、DoPa 網を使って、トラック上の Proxy と通常の Socket 通信を用いてデータのやりとりができる。

このような構成を採ることで、トラックとの通信手段が変化した場合でも、トラック Proxy を変更するだけですみ、Navimos-Master を変更する必要がない。

8.4 Nuts による Navimos-Master の開発

前節で説明した Win32/MFC 版の Navimos は、MFC フレームワークに、地図表示、音声合成、3D 表示などのブラックボックス部品を組み合わせて実装した。しかし、この実装では、ブラックボックス部品の拡張が不可能であるとともに、MFC フレームワークの Document-View 構造と Navimos 関連部品の結合関係が複雑であり、第三者による保守や柔軟な拡張が困難な構成となっていた。本節では、Nuts を用いたより柔軟な Navimos-Master の実装について説明する。Nuts 版 Navimos-Master の開発では、第 7.2節で説明した DRMA 地図ドライバを地図ビュアーとして再利用した。また、第 8.3.5節で説明した Win32 版 Navimos-Master のクラスを Nuts コンポーネントにカプセル化して再利用した。その上で、Nuts のグループマネジャを導入し、配送計画問題を一つの部品として扱えるようにしている。

8.4.1 Nuts 版 Navimos-Master の構成

Nuts 版 Navimos-Master の全体構成を図 8.13に示す。この中で、navimosManager はグループマネジャのサブクラスであり、部分木をグループ化して一つの配送計画部品として振舞わせる機能がある。navimosManager が管理するグループには、以下のものが含まれる。

- 地図ビュアー

- 地図縮尺変更コマンド
- 地図上の配送ノードアイコン
- 地図拡張のためのベクター (第 8.4.3参照)
- セービング法実行コマンド

navimosManager は、これらのグループに含まれる部品と以下のようなメッセージ通信を行う。

- NAVIMOS_SELECT_SAVING: セービング法実行コマンドが発するメッセージ。navimosManager は、このメッセージを受け取ると、セービング法アルゴリズムをカプセル化した部品を探索し、配車計画の立案を依頼する。次に、計画を地図に反映させるため、地図ビュアーに再描画メッセージを送る。

- NUTS_CLICK: 地図上のノードがクリックされたとき、ノード (正確にはノードに結合したベクター) から発せられる。navimosManager は、メッセージウインドウを探索して選択されたノードの名前などを表示する (第 8.4.3節でこの拡張機能ついて述べる)。

- NUTS_DCLICK: 地図上のノードがダブルクリックされたとき、ノード (正確にはノードに結合したベクター) から発せられる。navimosManager は、選択されたノードの詳細図を地図上にスーパーインポーズするように地図ビュアーにメッセージを送る (第 8.4.3節でこの拡張機能ついて述べる)。

- NUTS_DRAW: 外部からの要求に基づいて、地図ビュアーを再描画する。

- NUTS_ZOOMIN/OUT: 地図縮尺変更コマンドから発される拡大／縮小コマンドに応じて、地図ビュアーに縮尺変更メッセージを送る。

- NUTS_LOAD: 配送データをファイルから読み込み、地図ビュアーに反映させる。

- NUTS_OPEN: 外部からの要求に基づいて、自身を複製し、複製した部品に対して、NUTS_LOAD,NUTS_DRAW などのメッセージを送る。

新規に配送計画を行いたい場合には、既存の navimosManager 部品に NUTS_OPEN メッセージを送って複製する。複製を 1 回行った場合の全体構成を図 8.14に示す。複製された木の横幅が元のものよりも広いのは、複製後に、配送先ノードのデータを読み込み、それをアイコン部品としてウインドウ上に配置したためである。

このように Nuts 版 Navimos では、複数の配送計画部品を作成し、それぞれで異なった配送計画問題を解くことができる。プログラム上は、それぞれの配送計画を navimosManager をリーダーとする一つの部品として扱えるため、配送計画の新規作成／破棄が部品単位で簡単に行える。

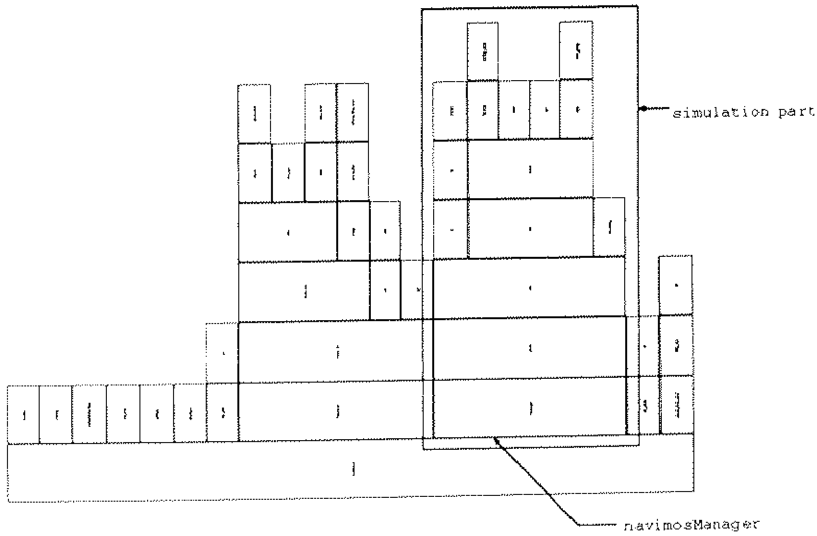

図 8.13: Nuts 版 Navimos の部品構成

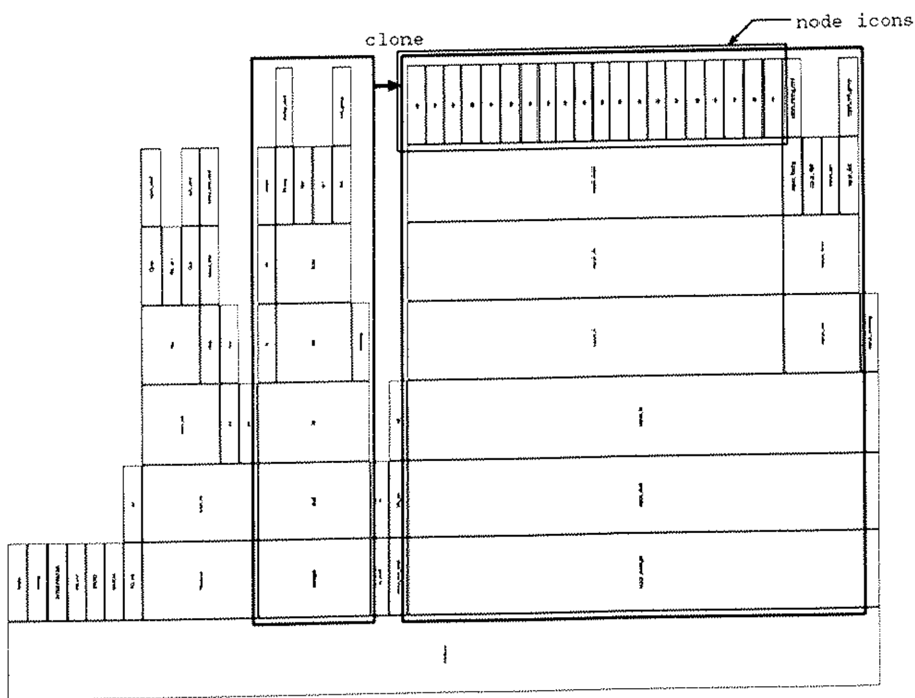

図 8.14: 複製した後の Nuts 版 Navimos の部品構成

```
newNutsXtApp(navimos);

newmapLoader(navimos,loader);//DRMA 地図データのローダー
newnavimosSaving(navimos,saving);//セービング法の部品
newNutsInterpreter(navimos,INTERPRETER);
newNutsStdOut(navimos,std_out);
newNutsListupObjectToPs(navimos,TO_PS);

newNutsXApplicationShell(navimos,Navimos);//メインウインドウの生成
newNutsFormWindow(Navimos,main_fm);
newNutsMenuBar(main_fm,main_mb);
newNutsPulldownMenu(main_mb,File);
newNutsPushButton(File,Open);
newNutsFileSelectionCommand(Open,open_cmd);
newNutsSeparator(File,file_sp1);
newNutsPushButton(File,Quit);
newNutsQuitCommand(Quit,quit_cmd);
newNutsPulldownMenu(main_mb,Help);
newNutsPushButton(Help,about_this);
newNutsAboutThisCommand(about_this,about_this_cmd);
newNutsScrolledText(main_fm,STDIO);
newNutsShellWindow(main_fm,sw);

newnavimosManager(navimos,manager);//navimosManagerグループの生成
newNutsXTransientShell(manager,shell);
newNutsFormWindow(shell,fm);
newNutsFrame(fm,fr);
newNutsDrawingAreaWindow(fr,da);
newmapvScopeWindow(da,scope);//DRMA 地図ビュアーの部品
newNutsMenuBar(fm,mb);
newNutsPulldownMenu(mb,Exec);
newNutsPushButton(Exec,Saving);
newnavimosExecSavingCmd(Saving,saving_cmd);//セービング法の実行コマンド
newNutsPushButton(Exec,TSP);
newNutsSeparator(Exec,sp1);
newNutsPushButton(Exec,Exit);
newNutsExitGroupCommand(Exit,exit_group);
newNutsSlotDial(fm,zooming);
newNutsXTransientShell(navimos,fs_shell);
newNutsFileSelectionBox(fs_shell,fs);

newNutsXTransientShell(navimos,about_this_shell);
newNutsAboutThisWindow(about_this_shell,alt);
```

図 8.15: Nuts 版 Navimos の構造記述ファイル

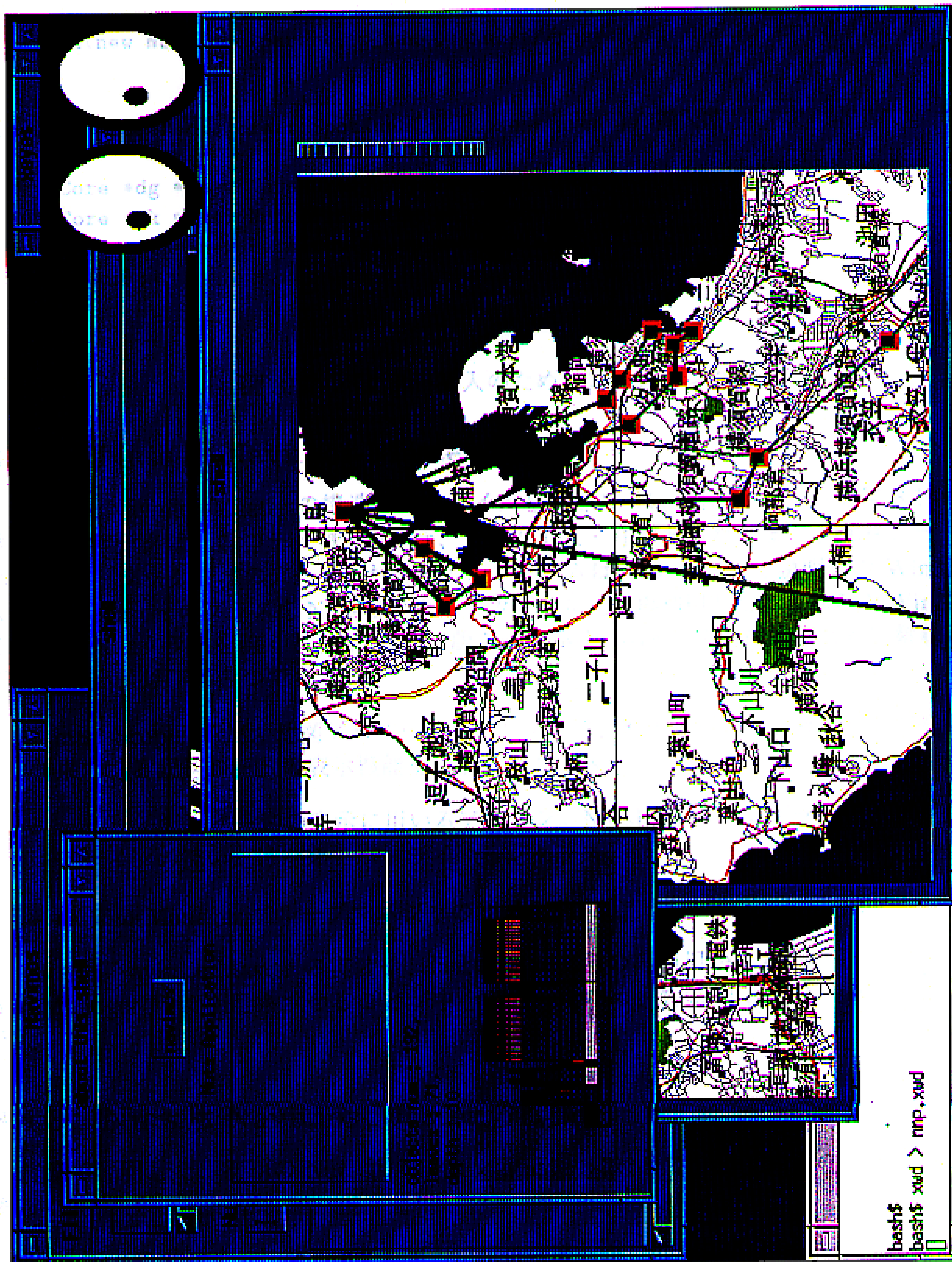

図 8.16: Nuts 版 Navimos の画面イメージ

```
NutsCore *nodeRec::_createSpotIcon(NutsCore *parent)//挿入前
{
  return(new NutsIcon(parent,sp_name,this));//parentは、ウインドウを表す
}

NutsCore *nodeRec::_createSpotIcon(NutsCore *parent)//挿入後
{
  NutsCore *dg = new NutsXDragger(parent,dg_name);
  NutsCore *pk = new NutsXPicker(dg,pk_name);
  NutsCore *sp = new NutsIcon(pk,sp_name,this);
  return(dg);
}
```

図 8.17: ベクター挿入のためのコーディング

8.4.2 Nuts 版 Navimos-Master の実装

Nuts 版 Navimos-Master の構造記述ファイルを図 8.15 に、画面イメージを図 8.16に示す。開発データは、総コード量 2913 行、工数 0.4 人・月であった。また、配送ノードやコース、トラックなどのデータ、およびアルゴリズムは、第 8.3.5節で説明した Win32 版 Navimos-Master のクラスを Nuts 部品としてカプセル化し再利用した。これらの Win32 版から移行したクラスを除くと、今回 Nuts 版 Navimos-Master の個別機能のために新規に開発した部品は以下の 3 つである。

- mapvScope:DRMA 地図ビュアーのサブクラスで、配送ノードや、セービング法の結果である配送ルートの表示機能を追加したクラス。
- spotIcon:配送ノードの表示に用いるアイコン。NutsIcon のサブクラスで、アイコンデザインの違いだけをオーバーライドしたクラス。
- navimosManager:グループマネジャのサブクラス。

Nuts 版 Navimos-Master の実装は、ほとんどが部品の再利用からなり、それらを navimosManager により連係させることが中心であったため、短時間 (0.4 人・月) で開発できた。

8.4.3 Nuts 版 Navimos-Master の拡張

前節で説明した、初期の Nuts 版 Navimos-Master に対して、以下のような拡張要求があった。

1. 地図上の配送ノードをドラグして、位置の微調整が可能なようにする— これにより、Navimos-Master は、経度緯度データの編集機能を持つ。Win32/MFC 版 Navimos では、経度緯度データを編集するためにダイアログボックスを使用していたが、地図上

のノードのドラグにより位置の微調整が行える方が、より扱いやすいインタフェースを提供できる。

2. 地図上の配送ノードのダブルクリックで近くに詳細地図をスーパーインポーズする— これにより、部分的な詳細情報の表示のために地図全体のスケールを変更する必要がない。Win32/MFC 版 Navimos では、別のウインドゥ (ツールバー) に拡大表示のための専用地図を用意していたが、地図上のノード近くにスーパーインポーズされる方が見やすい。

3. 詳細地図も地図上でドラグできる— 地図上に詳細地図をスーパーインポーズした場合に、スーパーインポーズされた地図に隠された部分の地図を表示させたい場合がある。そのため、スーパーインポーズされた地図がドラグにより別の場所に移動できるようにする。

1. の拡張については、アイコンをドラグするベクター (NutsDragger) を挿入してドラグ可能なようにした。また、2. の拡張については、アイコンを選択するベクター (NutsPicker) を挿入してアイコンのダブルクリックが検出できるようにした。これらのベクターの挿入は図 8.17に示すように、nodeRec クラス中のアイコン部品を作成するルーチンで行う (アイコン部品の生成は配送データに従って実行時に行われるので、構造記述ファイル中には記述できない)。

さらに、3. の拡張についても、詳細図を表示する Nuts コンポーネント (NutsIcon から派生して新規に開発) とその親ウインドゥの間にドラグのためのベクター (NutsDragger) を挿入することで実現した。

このように、ベクターを再利用することで、拡張機能を容易に実現できた。また、ベクターを Nuts の木構造に従って挿入するだけで、拡張機能の追加が簡単に行えた。

8.5 考察

現在の Win32 版 Navimos-Master は、第8.3.5節で説明したクラスのみを独自定義とし、フレームワークに関する部分は、MFC の Document-View アーキテクチャを用いている。すなわち、現在の Navimos クラス群の基底クラスは MFC の CObject クラスであり、NutsCore クラスではない。そのため、Nuts フレームワークに従った柔軟なコンポーネントの追加や変更、またコンポーネント差分プログラミングなどは行えない。また、フレームワークを MFC に委ねているため、MFC のコーディング規則を修得しなければ、Navimos の構造を理解することは難しい。今後、Win32API をカプセル化した Nuts コンポーネントを開発し、Navimos 全体を Nuts コンポーネントの積み上げで行えるようにする必要がある。

一方、Nuts 版 Navimos は現在 Unix 上でしか動作しないが (Nuts ライブラリが Unix 用のものしか存在しないため)、今後配送計画フレームワークを構築していく上で、以下のような利点がある。

- アルゴリズムやビューの種類の切替えなどを構造記述ファイルの編集によるコンポーネントの取り替えで容易に行えるようになる。コンポーネントを入れ換えて、性能を比較することでNavimosシステムを物流システム構築の実験ベンチとしても利用できる。
- 各コンポーネントはホワイトボックス拡張が可能であり、アルゴリズムやインタフェースの専門家によって、柔軟に新規クラスの開発が可能である。アルゴリズムやビューの開発とそれらのNavimosへの組み込み／保守の作業を分担して行うことができる。
- 物流の現場で使用される支援ソフトは、直観的でわかりやすいインタフェースが必要である。例えば、Navimos-Masterのドラグ&ドロップ操作による計画の修正機能などが必要とされる。このような機能をNutsのベクターコンポーネント群の再利用により容易に組み込める。
- Nutsコンポーネントをブラックボックス／ホワイトボックス両面から再利用することで、図8.1に示すようなNavimosシステム全体の開発を迅速に進めることができる。

以上のように、Navimosシステム全体をNutsコンポーネントの積み上げで構築する効果は大きい。一方、広く使われる実務システムとしては、Windows95上での動作は不可欠であり、今後Win32APIを採り入れたNutsコンポーネントを開発する必要がある。

本章で述べた、MFC版NavimosとNuts版Navimosの開発効率の比較については、第9.1.2節で述べる。