〔非公開〕

ＴＲ－Ｃ－００４１

通信ソフトウェア設計支援システム

島　健一　KENICHI SHIMA
佐藤　隆　TAKASHI SATOU
林　潔　KIYOSHI HAYASHI

１９９０．２．６

ＡＴＲ通信システム研究所

通信ソフトウエア設計支援システム

島 健一

1990年2月6日

もくじ

第 1 章

はじめに

本報告では、通信ソフトウェアの設計過程 (主に概要設計) における意志決定の知的な支援を行い、設計自動化に向けた上流工程での研究を進める上で重要となる試作ツールの概要についてのべる。

ソフトウェアの設計はきわめて知的な作業であり、高度な判断が要求される。たとえば、通信ソフトウェアの設計では、多数のプロセス間の相互作用を効率よく管理する必要がある。しかし、実際の設計にあたっては、それ以外にも、リアルタイム制御や多種多様なリソース管理 (ネットワーク、トラフィック)、高信頼性、高度な障害処理、セキュリティ管理を満たすソフトウェアを設計しなければならない。その際、通信手段、プロトコル制御方式、信号・データ変換手順、など決定すべき項目が多く、その決定過程の支援は重要である。これらをカードシステムをベースに開発した。

ここでは、通信ソフトウエアを開発する上で必要な、 SDL, Sequence chart の機能概要について述べ、それらを利用した設計判断履歴を用いたシステムの操作を示す。

また、これらの基盤となるカードシステムの機能も合わせて述べる。

最後に、本試作ツール開発の機会を与えて下さった山下 紘一社長、竹中 豊文室長、門田 充弘元室長 (現 NTT 交換研究所) に感謝します。

また、本試作システムに関して討論していただいた通信ソフトウェア研究室の各位に感謝します。

第 2 章

開発目的

2.1 開発目的

カードシステムを用いて設計過程の思考履歴、変更管理が行なえるシステムの開発を行ないました。

そのシステムの目的は設計過程のさまざまな情報 (特に判断に関するもの) をカードシステムを使用して記録・保存し、設計作業の結果としての SDL を自動的に生成することを目指します。

また判断の変更に伴う思考の修正や、蓄積された情報を再利用するための支援も行ないます。

2.2 システム概要

- 入力情報
  要求仕様書
- 参照情報
  通信世界に関する知識、ソフトウェア設計に関する知識
- 出力情報
  設計仕様書 (SDL)

2.3 システム構成

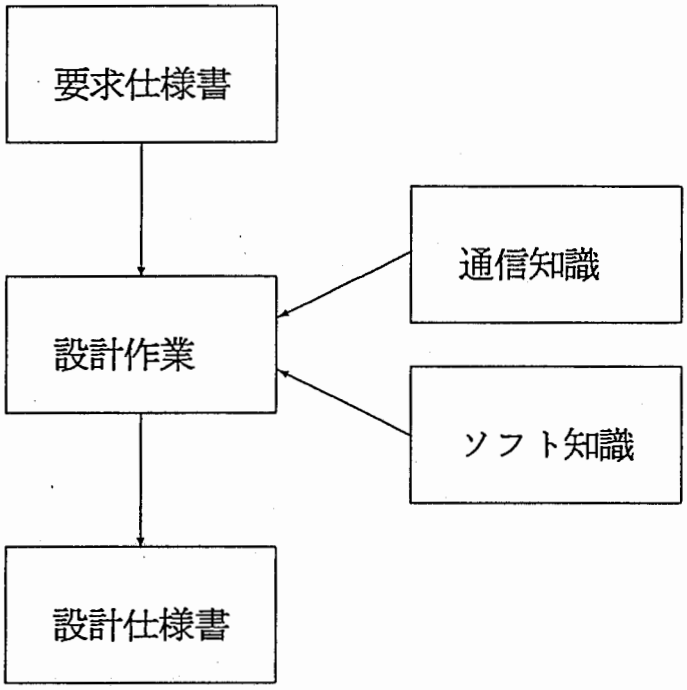


図 2.1: システム概要

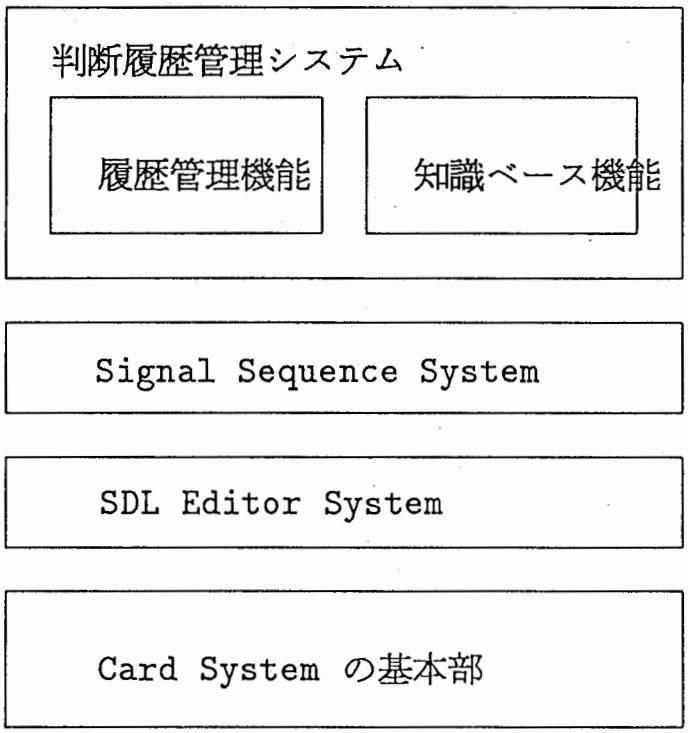


図 2.2: システム構成

第 3 章

DESCARTES の起動法

3.1 機能概要

本システムは、カードシステムを用いた設計過程の思考履歴、変更管理のシステムです。

このシステムの目的は設計過程のさまざまな情報 (特に判断に関するもの) をカードシステムを使用して記録・保存し、設計作業の結果としての SDL を自動的に生成することを目指ています。

また判断の変更に伴う思考の修正や、蓄積された情報を再利用するための支援も行ないます。

- 入力情報
  要求仕様書
- 参照情報
  通信世界に関する知識、ソフトウェア設計に関する知識
- 出力情報
  設計仕様書 (SDL)

現在のリリース・バージョンは 2.0 です。

3.2 保存場所

card-system のフィジカル・パス及びロジカル・パスは、次のとうりです。

1. フィジカル・パス
   (a) COM
   ```
   CLM01:>local>binary>descartes>
   ```
   (b) COM2
   ```
   CLM13:>local>binary>descartes>
   ```
2. ロジカル・パス
   ```
   card-system:descartes;
   ```

詳しくは、次のファイルを参照して下さい。

- sys;site;descares.system
- sys:site;card-system.translations

3.3 起動方法

*Dynamic Lisp Listener*で次のようにします。

Command: Load System *descartes*

次にカードシステム・マネージャを作成して下さい。

Command: Create Card System Manager *manager-name* :User Name *user* :Pathname *pathname* :System Type Descartes

Select A で、カードシステムのメイン・オペレーション・ウインドウが選択できます。

3.4 デモの起動方法

判断履歴管理システム (トレース付き) のデモのセット・アップ方法を以下に示します。
判断履歴管理システムのセットアップ方法

1. Login
   User descartes で Login します。

   command: login *descartes*

2. System Load
   clm02:>descartes>lispm-init.lisp で必要なファイルは全て Load されます。(約 15 分)

3. Data Load
   Select A でカードシステムの Main Operation Window を選択します。 Card System Menu (#\Hyper-Shift-Mouse-R で起動) の Control カラムの Attributes を選択します。

   以下の項目を次のようにセットします。

   ```
   Default pathname: CLM02:>descartes>card-system>demo-data>
   Default filename: decision-demo
   ```

   次に、 Card System Menu の Control カラムの Load を選択します。 これでデータがロードされます。(約 5 分)

   English : 英語版をロードする場合は、 Default filename: demo-english にします。

   データがロードされたあと、以下のフォームを評価して下さい。

   ```
   (dhc::add-decision-history-control-system-objects-to-card-system
                          card-system::*current-card-system-manager*
                          t)
   ```

4. Window Select
   Main Operation Window で Card System Menu の Link カラムの Select Window を選択し、 Link Window を作成します。次に Link Window を color-screen に移動させます。これは、 Card System Menu の Control カラムの Change Screen を使用します。

   次にデモフロー用の Window を表示させます。これは、 Main Operation Window の 判断履歴の蓄積と構造化 というカードアイコンを #\Mouse-L でクリックすることにより行えます。(マウスが color-screen 上にある時は、 System Menu (#\Shift-Mouse-R で起動) の Windows カラムの Set Mouse Screen でマウスを main-screen に移動させてから処理を行って下さい。)

Demo の方法

1. 初期画面
   初期画面は、次のようにして下さい。

   main-screen : 左側 観察実験の概要 Window、右側 判断履歴の蓄積と構造化 Window (判断履歴の蓄積と構造化 Window は上記の方法で表示させ、観察実験の概要 Window は 判断履歴の蓄積と構造化 Window の 観察実験の概要 アイコンを選択する事により表示されます。)

color-screen: 全面に プロトコル図の表示 Window を表示 (判断履歴の蓄積と構造化 Window の プロトコル図の表示アイコンを選択する事により表示されます。この時、 Window が main-screen に表示された場合は以下の用にして Window を color screen に移動させます。

(1) プロトコル図の表示 Window をセレクトする。

(2) typeout-window を表示させる。 (Suspend キ - を押 す)

(3) (send dw:*program-frame* :set-superior color:color-screen) という フォ - ムを評価する。これにより Window が color-screen に表示されます。

(4) typeout-window を解除させる。 (Abort 又は Resume キ - を押す)

(5) System Menu の This Window カラムの Expand で Window を全面に表示させる。

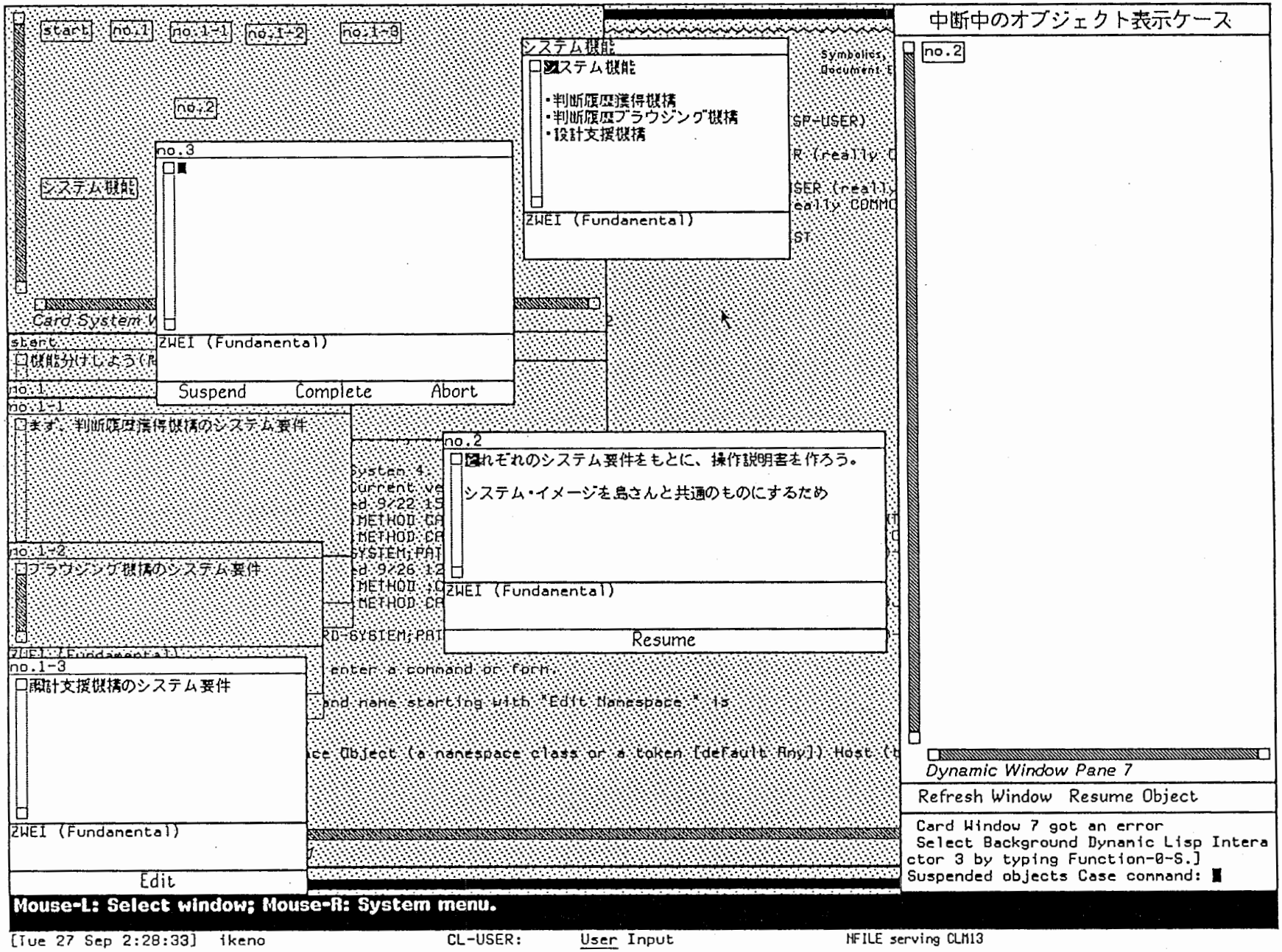
2. Demo の方法
デモは、 判断履歴の蓄積と構造化 Window 上のアイコンを #\Mouse-L でクリックすることにより該当する Window を表示させることにより行います。アイコンの項目は以下の通りです。

- 観察実験の概要
- 要求原案の表示
- プロトコル図の表示
- 候補、別解の作成
- 問題分割
- 設計過程の追跡
- 作成プログラムの表示

設計過程の追跡 以外は Window が表示されるだけです。

3. 設計過程の追跡
設計過程の追跡 アイコンをクリックすると Main Screen に判断カ - ド、プロダクトカ - ド の各 Window が次々表示され Color Screen には、判断履歴グラフ Window が表示されます。

このとき、判断カードが表示れた後に Menu が出ます。この Menu を選択 (何でもよいからマウスでクリックする) すると次の Window が表示されます。 Menu からマウスを離せばこれらの一連の処理が中止されます。


start
no.1
no.1-1
no.1-2
no.1-3
no.2
no.3
システム機能
・判断履歴獲得機構
・判断履歴ブラウジング機構
・設計支援機構
ZWEI (Fundamental)
Suspend
Complete
Abort
まず、判断履歴獲得機構のシステム要件
それぞれのシステム要件をもとに、操作説明書を作ろう。
システム・イメージを島さんと共通のものにするため
Resume
ブラウジング機構のシステム要件
設計支援機構のシステム要件
Edit
中断中のオブジェクト表示ケース
Dynamic Window Pane 7
Refresh Window
Resume Object
Card Window 7 got an error
Select Background Dynanic Lisp Intera
ctor 3 by typing Function-0-S.]
Suspended objects Case command:
Mouse-L: Select window; Mouse-R: System menu.
[Tue 27 Sep 2:28:33] ikeno
CL-USER:
User Input
NFILE serving CLM13

第 4 章

システムを構成するオブジェクト

4.1 システムを構成するオブジェクト

判断履歴管理システムを構成するオブジェクトを以下に述べる。

- 判断・カード (Decision Card)
  知識の状態 (State of Knowledge) を表すカードで、状態管理機能を持つ。
- 戻り位置管理ケース
  判断・カードを格納するケースで、戻り位置の選択時に使用される。
- 判断グラフ・リンク (Decision Graph Link)
  判断・カードを関係付るリンクで、話題の項目は、たて方向、話題の深さは、横方法で関係を表す。
- プロダクト・カード (Product Card)
  SDL、信号シーケンス図、設計仕様書等。
- プロダクト・リンク
  判断・カードとプロダクト・カードを関係付けるリンクである。
- 問題分割リンク
  一つの判断カードを分割して新たに作成された判断カードを関係付けるリンクである。
- 別解リンク
  一つの判断カードから別解案を幾つか作成し、その判断カードを関係付けるリンクである。

ユーザは上記のオブジェクトとカード・システムの他のベーシック・オブジェクトを作成しながら設計作業を進める。

第 5 章

判断カード

5.1 判断・カードの内容

判断・カードに記入する内容を以下に示す。

- 設計者
- 対象 (何について)
- 決定方法 (どうやって)
- 参考資料 (インプット)
- 決定事項 (どう決めたか)
- 下心情報
- 判断強度 (自信)
- 課題

5.2 判断・カードの状態管理

判断・カードで管理する状態とその状態変更コマンドを以下に示す。
判断・カード (状態管理カード) から、思考の状況 (suspend、 complete、 abort) に応じて、判断履歴を獲得する。

- suspend(中断)

  - より緊急な検討事項が発生した。
    次の新しいオブジェクトを作成し、問題行動グラフ・リンクの話題項目をセットする。自分自身を戻り位置管理ケースに登録する。
  - 現在の思考が他と比べて深くなりすぎた。
    次のオブジェクトを新しく作成するか、または戻り位置管理ケースから選択するのかを選択する。
    新しいオブジェクトを作成する手順及びセット項目は上記と同様である。
  - 不確定要素が多く、思考対象が多すぎる。
    同上。

- 思考結果が既存の知識と一致または矛盾する。
  戻り位置に復帰する。(位置管理ケースの先頭)
- 話題が本筋から外れた。
  戻り位置に復帰する。(位置管理ケースの先頭)

- complete(完了)

  - 目的としたレベルまで推論した。
    新しくオブジェクトを作成する。(手順等は同上)
    問題行動グラフ・リンクは話題の深さをセットする。

- abort(中止)

  - 現在の思考が他と比べて深くなりすぎた。
    同上。
    問題行動グラフ・リンクは話題項目をセットする。
  - 思考結果が役に立たない。
    戻り位置に復帰する。(位置管理ケースの先頭)
  - 話題が本筋から外れた。
    戻り位置に復帰する。(位置管理ケースの先頭)

5.3 判断・カードのウインドウ・メニュー

判断・カードのウインドウ・メニューには、デフォルトのメニュー項目以外に次のメニュー・コマンドが割り当てられています。

- プロダクト選択
  判断・カードに関係付けられたプロダクトのウインドウをセレクトします。この時プロダクトが存在しない時は、次のプロダクト選択メニューから選んで作成します。
  - ベーシック・カード
  - SDL GR
  - Signal Sequence
- 問題分割
  判断・カードの内容を問題分割し、分割案の判断・カードを作成します。この時選択されていた判断・カードは中断状態になり、戻り位置管理ケースに登録されます。
- 別解
  判断・カードの内容の別解を作成し、別解案の判断・カードを作成します。この時選択されていた判断・カードは中断状態になり、戻り位置管理ケースに登録されます。
- 中断理由表示
  判断・カードの中断理由の履歴を表示します。

5.4　判断・カード内容の差分管理

思考の状況に応じて判断・カードの内容を差分管理する。

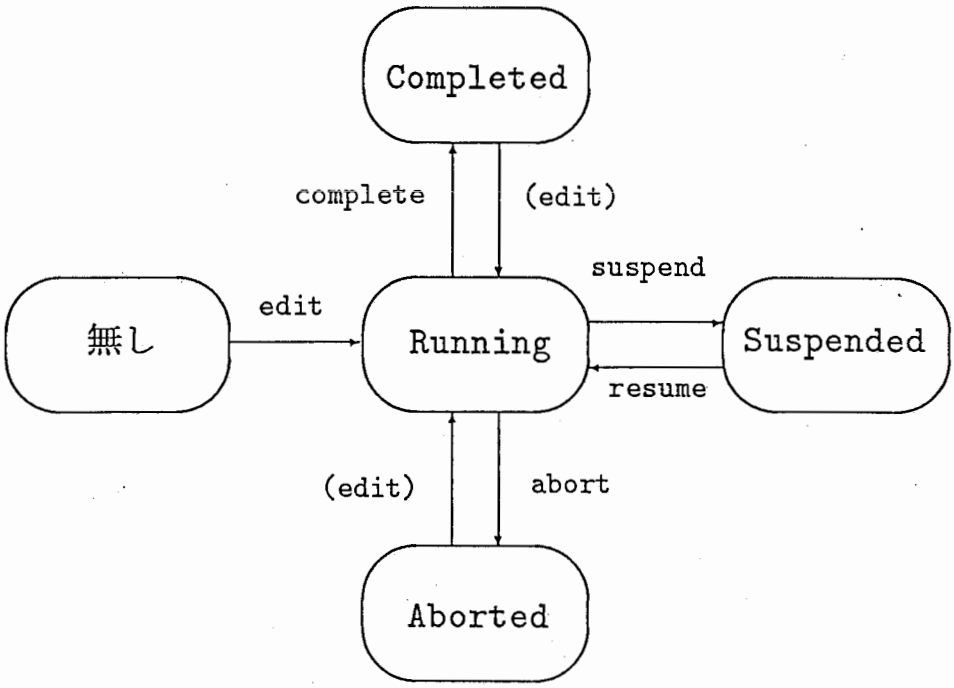


図 5.1: 判断カードの状態遷移

第 6 章

判断グラフリンク

6.1 判断グラフの作成 (表示)

判断グラフ・リンクは、判断グラフを作成するために必要な情報を属性として持つ。

- 話題の項目 (たて方向)
- 話題の深さ (横方向)

上記以外の属性としてラベルをセットすることができる。

画面をある一定の大きさの格子に分割し、判断・カードをノード、判断グラフ・リンクをアークとして上記の属性情報から判断行動グラフを作成し表示する。

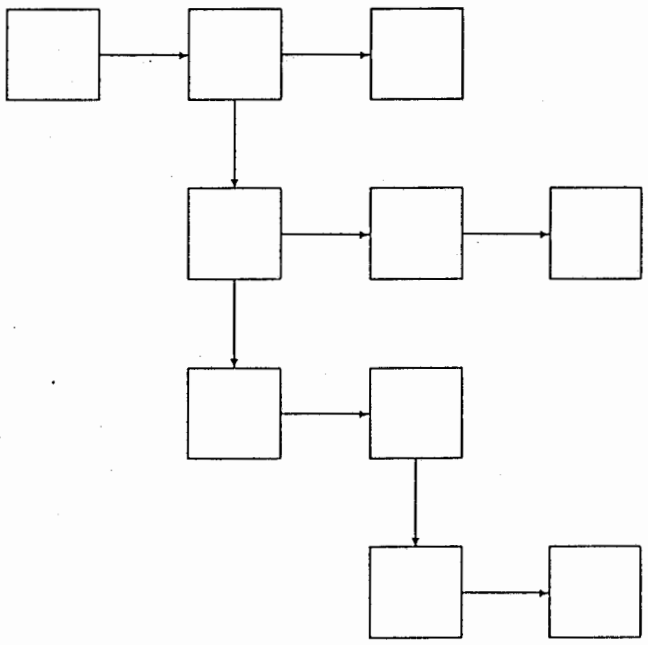


図 6.1: 問題行動グラフ

また、問題行動グラフ・リンク・ノードは判断・カードの状態により、以下の色で表示される。

- Completed
  Green
- Suspended
  Magenta
- Aborted
  Cyan
- Running
  Red

第 7 章

プロダクト・カードとリンク

7.1 プロダクト・カードの作成と選択

判断・カードのウインドウ・メニューから”プロダクト選択”を選ぶことによりプロダクト・カードを選択 (存在しない時は作成する) することができる。

プロダクト・カードは当面以下のものを考える。

- SDL-GR
- Signal Sequence
- Basic-Card(設計仕様書などのテキスト)

7.2 プロダクト・カード内容の差分管理

プロダクト・カードについても判断・カードと同様に、内容を差分管理する。

7.3 プロダクト・リンク

上記の手順で作成されたプロダクト・カードと判断・カードにプロダクト・リンクが自動的にセットされる。

第 8 章

問題分割

8.1 問題分割

判断・カードの分割したい部分にリージョンを指定し、ウインドウのメニューより“問題分割”を選択する。
新しくふたつのオブジェクトを作成し、そのオブジェクトに問題分割・リンクをセットする。

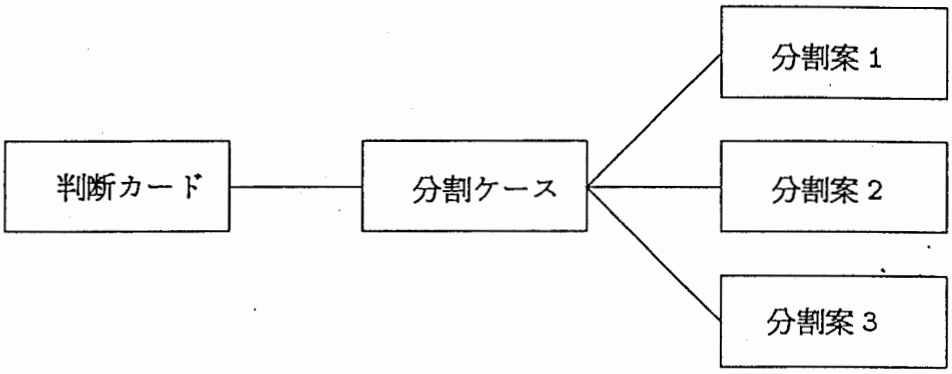


図 8.1: 分割リンク

作成されたオブジェクトに判断グラフ・リンクも同時にセットする。

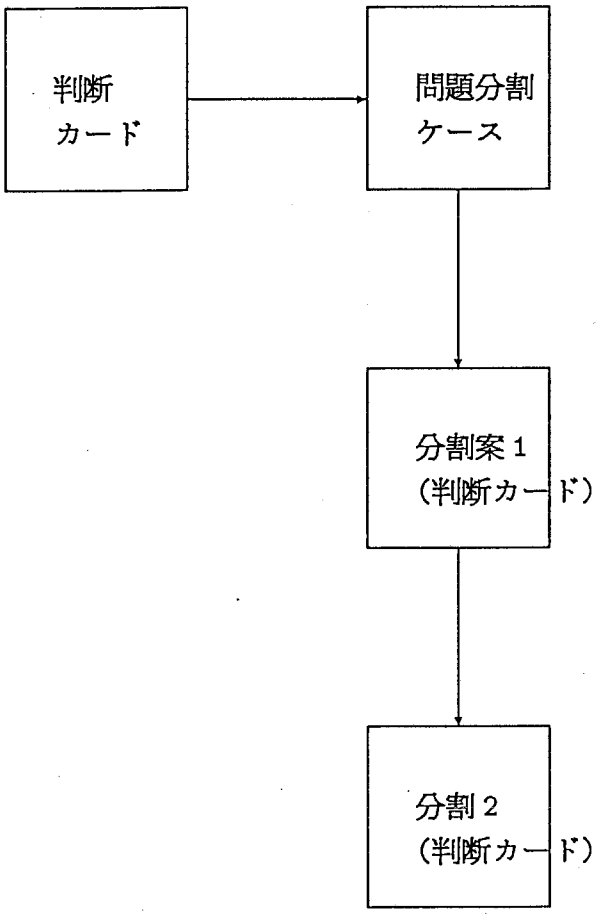


図 8.2: 問題行動グラフリンク

第 9 章

別解

9.1 別解

ウインドウのメニューより“別解”を選択する。新しいオブジェクトを作成して、別解・リンクをセットする。

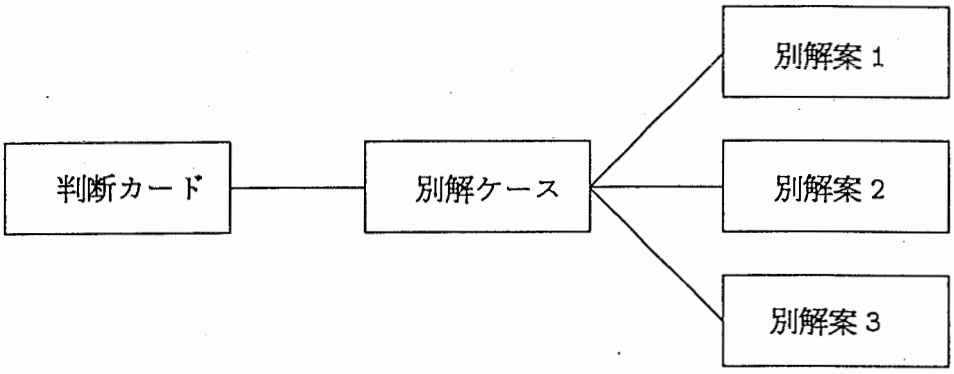


図 9.1: 分割リンク

作成されたオブジェクトに判断グラフ・リンクも同時にセットする。

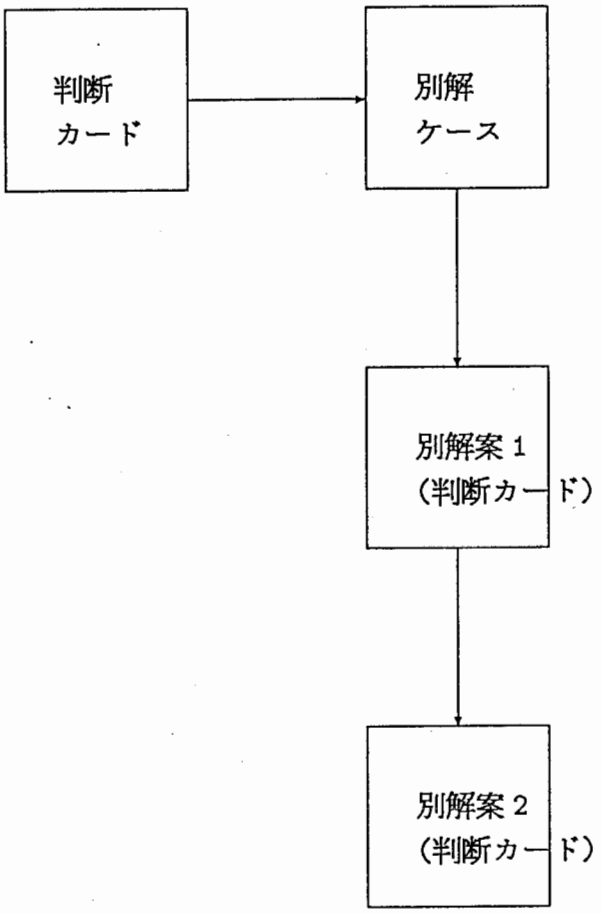


図 9.2: 判断グラフリンク

第 10 章

付録 通信ソフトウエア基本カードシステム

10.1 機能概要

従来のカード型のシステムでは主に情報の整理を目的に作成されていますが、ここでは次のような特徴を持ったカード型のデータベース管理システムが、カードシステムです。

特徴

- 情報と情報の関係を管理するリンクという概念を考え、そのリンクを階層的に管理することができる。
- カードの状態 (実行中、中断、中止、完了) を管理することができる。
- オブジェクト間に論理関係を定義することができる。
- 実行するオブジェクトを定義することができる。
- オブジェクト指向の考えに基ずきユーザーが自由にオブジェクト操作を行える様にする。

現在のリリース・バージョンは 1.4 です。

10.2 保存場所

card-system のフィジカル・パス及びロジカル・パスは、次のとうりです。

1. フィジカル・パス
   (a) COM
   `CLM01:>local>binary>card-system>rel-1-4>`
   (b) COM2
   `CLM13:>local>binary>card-system>rel-1-4>`
2. ロジカル・パス
   `card-system:card-system;`

詳しくは、次のファイルを参照して下さい。

- sys;site;card-system.system
- sys:site;card-system.translations

10.3 起動方法

*Dynamic Lisp Listener*で次のようにします。

Command: Load System *card-system*

次にカードシステム・マネージャを作成して下さい。

Command: Create Card System Manager *manager-name* :User Name *user* :Pathname *pathname* :System Type *system-type*

Select A で、カードシステムのメイン・オペレーション・ウインドウが選択できます。

第 11 章

基本仕様

11.1 カードシステムのオブジェクト

カードシステムには、以下の二つのタイプのオブジェクトが有ります。

- マネージャー
  カードシステムはひとつのホストで同時に複数のデータを編集することができます。ひとつひとつのデータの管理等をマネージャーが行います。
- 一般のオブジェクト
  カード、ケース、リンクという三つのオブジェクトが有ります。

実行中のマネージャーは、 card-system::*current-card-system-manamer* にバインドされています。

11.2 カードシステム・マネージャー

11.2.1 マネージャーの作成

"Create Card System Manager" コマンドを使用して作成します。コマンドの引き数は以下のものがあります。

manager name	マネージャーのタイトル
&key	
user name	ユーザー名 (default login user)
pathname	セーブするファイルのディレクトリ

11.2.2 マネージャーの変更

"Select Card System Manager" コマンドを使用して変更を行います。

11.2.3 マネージャーの初期化

カードシステムの旧バージョンのデータは、一旦マネージャを作成後データをロードして、 "Initialize Card System Manager Parameter" コマンドを使用してマネージャの初期化を行なって下さい。

11.3 ベーシック・オブジェクト

11.3.1 デフォルト・ベーシック・オブジェクト

カードシステムには、デフォルトで次のベーシック・オブジェクトが定義されています。

- ベーシック・カード
  カードの内容 (ストリング) を zwei:standalone-editor-frame を使用して編集を行ないます。
- ベーシック・状態管理・カード
  状態管理機能を持ったカードです。
  (編集方法はベーシック・カードと同じ)
- ベーシック・ケース
  オブジェクトをストァした逆順 (一番最後にストァされたものが先頭) に表示する。
- 中断中オブジェクト管理・ケース
  状態管理機能を持ったオブジェクトの中で、中断状態のオブジェクトがストァされます。
- ゴミ箱・ケース
  このケースにストアされたオブジェクトはセーブの対象になりません。
- ベーシック・グループ・リンク
  リンクの形式が単なるグループである。
- ベーシック・ツリー・リンク
  リンクの形式がツリー構造をしている。
- ベーシック・ネットワーク・リンク
  リンクの形式がネットワーク構造をしている。

11.4 カードシステムのウインドウ

カードシステムのウインドウは以下の四つが有ります。

- メイン・オペレーション・ウインドウ
  オブジェクトのアイコンを表示するウインドウ
- リンク・オペレーション・ウインドウ
  リンクの一覧表及び各リンクの内容を表示するウインドウ
- サーチ・オペレーション・ウインドウ
  カードシステムのオブジェクトを検索するウインドウ
- オブジェクト・ウインドウ
  オブジェクトの内容を表示するウインドウ

図 11.1: メイン・オペレーション・ウインドウ

図 11.2: リンク・オペレーション・ウインドウ

図 11.3: サーチ・オペレーション・ウインドウ

11.5 オブジェクトのスクリプト

スクリプトはオブジェクトの動作毎に指定することができます。指定方法は以下のとうりです。
(動作　　実行する関数　　{タイプ})

- 実行する関数の引き数
  動作する関数によって引き数の数は異なりますが一般的に以下のとうりです。

  第一引き数　　オブジェクト自身 (self)
  第二引き数以降　動作する関数の引き数の並び

- タイプについて
  タイプは以下のものが指定できます。

  :before　　標準動作を実行する前に行う処理
  :primary　標準動作以外の処理
  :after　　標準動作を実行した後に行う処理

  省略値は　:primary　です。

第 12 章

操作説明

12.1 起動と終了

12.1.1 起動

カードシステムは、Select A でメイン・オペレーション・ウインドウを選択できます。

システムをロードするだけでは、マネージャが作成されませんので、“Create Card System Manager” コマンドでマネージャを作成して下さい。

title	“card-system”
user-name	login user
default-pathname	login directory + card-system-data

12.1.2 終了

カードシステム・システムメニューの “Quit” を選択して下さい。

12.2 コマンド

コマンドには各ウインドウで共通に使用できるものと、特定のウインドウだけで使用できるものが有ります。

コマンドは原則としてマウス・オペレーションが行える様になっています。マウス・オペレーションが定義されていないコマンドは、“Card System Operation Menu” で選択することができます。(メニューは hyper-super-mouse-right で表示されます。)

12.2.1 共通コマンド

- Edit Attributes (meta-mouse-left)
  オブジェクトの属性を変更する。
- Edit Script (meta-mouse-middle)
  オブジェクトのスクリプトを変更する。
- Delete object (control-mouse-left)
  オブジェクトを削除する。

- Edit icon condition (meta-shift-mouse-left)
  アイコンの属性を変更する。

- Open object (mouse-letf)
  オブジェクトの内容を表示するウインドウを作成する。

- Resume object (mouse-letf)
  中断状態のオブジェクトを編集可能にする。

- Store object to case (shift-mouse-left)
  オブジェクトをケースにストァする。
  (ケースをマウスで指示する。)

- Store object to case from a menu
  オブジェクトをケースにストァする。
  (ケースをメニューから選択する。)

- Remove Object From Case (control-shift-mouse-left)
  オブジェクトをケースから削除する。
  (オブジェクトをマウスで指示する。)

- Remove Object From Case from a menu
  オブジェクトをケースから削除する。
  (オブジェクトをメニューで選択する。)

- Set Link to object (control-mouse-middle)
  オブジェクトにリンクをセットする。
  (リンクをメニューから選択し、関連するオブジェクトをマウスで指示する。)

- Set Link to object from a menu
  オブジェクトにリンクをセットする。
  (リンクをメニューから選択し、関連するオブジェクトをメニューから選択する。)

- Remove Object From Link (control-shift-mouse-left)
  リンクからオブジェクトを削除する。
  (オブジェクトをマウスで指示する。)

- Remove Object From Link from a menu
  リンクからオブジェクトを削除する。
  (オブジェクトをメニューで選択する。)

12.2.2 各ウインドウのコマンド

1. メイン・オペレーション・ウインドウ
   コマンド・メニュー

   - Expose Window
     オブジェクト・ウインドウを全て表示する。

- Refresh Window
  ウインドウをリフレッシュする。

コマンド

- Create Card (mouse-left)
  カードのオブジェクトを新しく作成する。
- Create Case (mouse-middle)
  ケースのオブジェクトを新しく作成する。
- Move Card (shift-mouse-middle)
  カードのアイコンを移動する。
- Move Case (shift-mouse-middle)
  ケースのアイコンを移動する。

2. リンク・オペレーション・ウインドウ
   コマンド・メニュー

- Show Link
  リンクの一覧表を表示する。
- Refresh Window
  ウインドウをリフレッシュする。

コマンド

- Show Link (mouse-left)
  リンクの内容を表示する。
- Add Connection To Link (control-mouse-middle)
  オブジェクトにリンクをセットする。
  (オブジェクトをマウスで指示する。)
- Add Connection To Link from a menu
  オブジェクトにリンクをセットする。
  (オブジェクトをメニューで選択する。)
- Remove Connection From Link (control-mouse-left)
  オブジェクトのリンクを削除する。
  (オブジェクトをマウスで指示する。)
- Remove Connection From Link from a menu
  オブジェクトのリンクを削除する。
  (オブジェクトをメニューで選択する。)

3. サーチ・オペレーション・ウインドウ

コマンド・メニュー及びコマンド

- Store Contents
  検索結果をケースにストアする。

- Search　Keyword
  オブジェクトをキーワードで検索する。
- Search　Object
  オブジェクトの内容をストリングで検索する。
- Reset　Contents
  検索結果をクリアする。

12.3 メニュー

カードシステムには、次の二つのメニューが有ります。

- カードシステム・システムメニュー
  このメニューはカードシステムの全てのウインドウで　hyper-shift-mouse-right　により起動させることができます。
- オブジェクト・ウインドウ・メニュー
  このメニューはオブジェクトのウインドウで　hyper-mouse-right　により起動させることができます。

図 12.1: カードシステム・システムメニュー

図 12.2: オブジェクト・ウインドウメニュー

12.3.1 カードシステム・システムメニュー

このメニューは次の四つのカラムで構成されています。

- カード・カラム
  カードのオブジェクトに対する操作を行ないます。
- ケース・カラム
  ケースのオブジェクトに対する操作を行ないます。

- リンク・カラム
  リンクのオブジェクトに対する操作を行ないます。
- システム・コントロール・カラム
  システム全体の操作を行います。

1. カード・カラム

   - Define
     新しいフォーム・カードを定義します。
   - Create
     カードを新しく作成します。
   - Select
     カード・ウインドウを選択します。
   - Atttributes
     カードの属性を変更します。
   - Icon Condition
     カード・アイコンの属性を変更します。
   - Edit Script
     カードのスクリプトを変更します。

2. ケース・カラム

   - Create
     ケースを新しく作成します。
   - Select
     ケース・ウインドウを選択します。
   - Atttributes
     ケースの属性を変更します。
   - Icon Condition
     ケース・アイコンの属性を変更します。
   - Edit Script
     ケースのスクリプトを変更します。

3. リンク・カラム

   - Define
     新しいリンクを作成します。
   - Select Window
     リンク・ウインドウを選択します。

- Search
  リンクの検索を行ないます。
- Atttributes
  リンクの属性を変更します。
- Edit Script
  リンクのスクリプトを変更します。

4. システム・コントロール・カラム

- Load
  カードシステムのオブジェクト・データを読み込みます。
- Atttributes
  マネージャの属性を変更します。
- Edit Script
  マネージャのスクリプトを変更します。
- Select Suspended
  中断中オブジェクト管理ケースを選択します。
- Select Running
  実行中のオブジェクト・ウインドウを選択します。
- Select Trash
  ゴミ箱・ケースを選択します。
- Search
  オブジェクトの検索を行なうサーチ・ウインドウを選択します。
- Group
  オブジェクトにグループ付けを行います。
- Undo
  直前のコマンドを取り消します。
- Save
  カードシステムのオブジェクト・データを保存します。
- Quit
  カードシステムを終了します。
- Help
  HELP・ウインドウを作成します。
- Change Screen
  カード・システムに関わるウインドウを、メイン・スクリーンからカラー・スクリーン又はその逆に移動させます。

12.3.2 ベーシック・オブジェクト・ウインドウ・メニュー

- Close
  オブジェクト・ウインドウを削除します。

- Copy
  オブジェクトをコピーします。
- Store
  オブジェクトをケースしストァします。
- Atttributes
  オブジェクトの属性を変更します。
- Script
  オブジェクトのスクリプトを変更します。
- Delete
  オブジェクトを削除します。
- Set　Link
  オブジェクトにリンクをセットします。

第 13 章

インストレーション

カードシステムはシステムとして定義されていますから、次のコマンドを発行することにより、利用することが可能になります。

Command: Load System Card-System

(詳細は、 “sys:site;card-system.system”
“sys:site;card-system.translations”
を参照して下さい。)

第 14 章

デフォルト・オブジェクトのスクリプト

14.1 basic-card

:open-self	内容を編集するウインドウをオープンする (self)
:make-window	内容を編集するウインドウを作成する (self &rest options &key :superior &allow-other-keys)
:close-self	内容を編集するウインドウをクローズする (編集後の内容は保存される) (self)
:choose-menu-item	オブジェクト・ウインドウのメニュー項目を求める (self)
:move-self	メイン・ウインドウ上のアイコンを移動する (self new-x new-y)
:present-self	自分自身をどうみせるか (self stream &key window-type object &allow-other-keys)
:present-icon	自分自身をどうみせるか (self stream &key window-type object &allow-other-keys)
:delete-self	削除 (self)
:add-case	ケースに追加する (self case)
:add-link	リンクに追加する (self link)
:remove-case	ケースから削除する (self case)
:remove-link	リンクから削除する (self link)

14.2 basic-status-control-card

:open-self	内容を編集するウインドウをオープンする
	(self)
:make-window	内容を編集するウインドウを作成する
	(self &rest options &key :superior &allow-other-keys)
:close-self	内容を編集するウインドウをクローズする
	(編集後の内容は保存される)
	(self)
:choose-menu-item	オブジェクト・ウインドウのメニュー項目を求める
	(self)
:move-self	メイン・ウインドウ上のアイコンを移動する
	(self new-x new-y)
:present-self	自分自身をどうみせるか
	(self stream &key window-type object &allow-other-keys)
:present-icon	自分自身をどうみせるか
	(self stream &key window-type object &allow-other-keys)
:delete-self	削除
	(self)
:add-case	ケースに追加する
	(self case)
:add-link	リンクに追加する
	(self link)
:remove-case	ケースから削除する
	(self case)
:remove-link	リンクから削除する
	(self link)
:edit	編集
	(self)
:suspend	中断
	(self)
:complete	完了
	(self)
:abort	中止
	(self)
:resume	継続
	(self)

14.3 basic-case

:open-self	内容を編集するウインドウをオープンする
	(self)
:make-window	内容を編集するウインドウを作成する
	(self &rest options &key :superior &allow-other-keys)
:close-self	内容を編集するウインドウをクローズする
	(self)
:choose-menu-item	オブジェクト・ウインドウのメニュー項目を求める
	(self)
:move-self	メイン・ウインドウ上のアイコンを移動する
	(self new-x new-y)
:present-self	自分自身をどうみせるか
	(self stream &key window-type object &allow-other-keys)
:present-icon	自分自身をどうみせるか
	(self stream &key window-type object &allow-other-keys)
:add-contents	内容を追加する
	(self object)
:remove-contents	内容を削除する
	(self object)
:present-contents	内容をどうみせるか
	(self stream)
:delete-self	削除
	(self)
:add-case	ケースに追加する
	(self case)
:add-link	リンクに追加する
	(self link)
:remove-case	ケースから削除する
	(self case)
:remove-link	リンクから削除する
	(self link)

14.4 suspended-object-store-case

:open-self	内容を編集するウインドウをオープンする (self)
:make-window	内容を編集するウインドウを作成する (self &rest options &key :superior &allow-other-keys)
:close-self	内容を編集するウインドウをクローズする (self)
:choose-menu-item	オブジェクト・ウインドウのメニュー項目を求める (self)
:move-self	メイン・ウインドウ上のアイコンを移動する (self new-x new-y)
:present-self	自分自身をどうみせるか (self stream &key window-type object &allow-other-keys)
:present-icon	自分自身をどうみせるか (self stream &key window-type object &allow-other-keys)
:add-contents	内容を追加する (self object)
:remove-contents	内容を削除する (self object)
:present-contents	内容をどうみせるか (self stream)

14.5 trash-case

:open-self	内容を編集するウインドウをオープンする
	(self)
:make-window	内容を編集するウインドウを作成する
	(self &rest options &key :superior &allow-other-keys)
:close-self	内容を編集するウインドウをクローズする
	(self)
:choose-menu-item	オブジェクト・ウインドウのメニュー項目を求める
	(self)
:move-self	メイン・ウインドウ上のアイコンを移動する
	(self new-x new-y)
:present-self	自分自身をどうみせるか
	(self stream &key window-type object &allow-other-keys)
:present-icon	自分自身をどうみせるか
	(self stream &key window-type object &allow-other-keys)
:add-contents	内容を追加する
	(self object)
:remove-contents	内容を削除する
	(self object)
:present-contents	内容をどうみせるか
	(self stream)

14.6 basic-group-link

:present-self	自分自身をどうにせるか
	(self stream &key window-type object &allow-other-keys)
:delete-self	削除
	(self stream)
:present-contents	内容をどうみせるか
	(self stream)
:add-connection	内容の追加
	(self object)
:remove-connection	内容の削除
	(self object)
:remove-object	オブジェクトの削除
	(self object)

14.7 basic-tree-link

:present-self	自分自身をどうみせるか
	(self stream &key window-type object &allow-other-keys)
:delete-self	削除
	(self)
:present-contents	内容をどうみせるか
	(self stream)
:add-connection	内容の追加
	(self root node)
:remove-connection	内容の削除
	(self root node)
:remove-object	オブジェクトの削除
	(self object)

14.8 basic-network-link

:present-self	自分自身をどうみせるか
	(self stream &key window-type object &allow-other-keys)
:delete-self	削除
	(self)
:present-contents	内容をどうみせるか
	(self stream)
:add-connection	内容の追加
	(self object1 object2)
:remove-connection	内容の削除
	(self object1 object2)
:remove-object	オブジェクトの削除
	(self object)

14.9 card-system-manager

:load-object	データのロード
	(self)
:save-object	データのセーブ
	(self)
:new-card 新しいカードの作成	
	(self options)
:new-case	新しいケースの作成
	(self options)
:new-link	新しいリンクの作成
	(self options)
:delete-card カードの削除	
	(self card)
:delete-case	ケースの削除
	(self case)
:delete-link	リンクの削除
	(self link)
:change-status-after	実行中の状態管理オブジェクトの状態が変更されたあとの処理
	(self object status)
:call-card-system-menu	カード・システム・メニューの表示
	(self window)
:call-object-window-menu	オブジェクト・ウインドウ・メニューの表示
	(self window)

第 15 章

検索関数

15.1 オブジェクト検索関数

オブジェクトを検索するために次の関数があります。

1. タイトルによる検索
   card-system::find-object-from-title

title	オブジェクトのタイトル (ストリング)
&key	
:type	オブジェクトのタイプ
	(:all,:card,:case,:link default=:all)
:superior	オブジェクトの属すマネージャー
	(default=card-system::*current-card-system-manager*)

2. 一連番号による検索
   card-system::find-object-from-unique-id

unique-id	オブジェクトの一連番号
&key	
:type	オブジェクトのタイプ
	(:card,:case,:link default=:card)
:superior	オブジェクトの属すマネージャー
	(default=card-system::*current-card-system-manager*)

3. ウインドウによる検索
   card-system::find-object-from-window

window	ウインドウ

15.2 マネージャー検索関数

マネージャーを検索するために次の関数があります。

1. オブジェクトによる検索
   card-system::find-manager-from-object

object　オブジェクト

2. ウインドウによる検索
   card-system::find-maanger-from-window

window　ウインドウ

15.3　ウインドウ・タイプ検索関数

ウインドウ・タイプを検索するために次の関数があります。

1. ウインドウによる検索
   card-system::find-window-type-from-window

window　ウインドウ

第 16 章

スクリプトの例

16.1 カードシステム スクリプト例

スクリプトの例を示す。これは以下のファイルの内容です。

card-system:card-system;examples;script-examples.lisp.newest

```
;;; -*- Mode: LISP; Syntax: Common-lisp; Package: USER; Base: 10 -*-

;;; card

;;; :open-self 内容を表示するための window を作成し select する。
;;;
;;;  window の編集モードを lisp-mode にする。

(:open-self
  (lambda (self)
    (let* ((window (send self :window-object))
   (buffer (send window :io-buffer)) )
      (tv:io-buffer-put buffer card-system::*lisp-mode-char*) ))
 :after)


;;;  window の編集モードを text-mode にする。

(:open-self
  (lambda (self)
    (let* ((window (send self :window-object))
   (buffer (send window :io-buffer)) )
      (tv:io-buffer-put buffer card-system::*text-mode-char*) ))
 :after)

;;; case
```

```
;;; :add-contents,:remove-contents
;;;   ケースにオブジェクトがストァされたあとの処理、ケースからオブジェクトが削除されたあとの処理。
;;;
;;;   ケースにオブジェクトがストァされたあと、オブジェクトを main operation window に表示しないようにする。
;;;
;;;   :add-contents
;;;    main operation window の表示フラグをオフにする。
;;;
;;;   :remove-contents
;;;    削除されたオブジェクトがどのケースにも含まれないとき、
;;;    main operation window の表示フラグをオンにする。

(:add-contents (lambda (self object)
 ;; 使用しない
 (ignore self)
 ;; object の status を変更する
 (send object :set-status nil))
 :after)

(:remove-contents (lambda (self object)
    ;; 使用しない
    (ignore self)
    ;;; どのケースにも属さないとき
    (unless (send object :cases)
      ;; object の status を変更する
      (send object :set-status t) ))
 :after)

;;; :present-contents ケースの内容を表示する。
;;;
;;;  ケースの window に内容以外のものを表示する。

(:present-contents (lambda (self stream)
     (fresh-line stream)
     ;; ケースのオブジェクトのコメントを表示する
     (write-string (send self :comment-string) stream) )
 :before)

;;;  ケースの内容を編集して表示する。
;;;   タイトルの文字順にソートする。

(:present-contents (lambda (self stream)
```

```
    (let ((contents-l
    ;; タイトルとオブジェクトとの対応リストを作成する
    (loop for object in (send self :contents)
  collect (list (send object :title) object) )))
      ;;(fresh-line stream) 使用してはならない
      (formatting-table (stream :inter-row-spacing 3)
;; タイトルの文字順にソートする
(dolist (item (sort contents-l #'string< :key #'car))
  (formatting-row (stream)
    (formatting-cell (stream)
      (card-system::present-self (second item)
 stream
 ;; 次の引数が追加された
 :window-type :case
 :object self) ))))))
)

;;;  ケースにオブジェクトがストァされるとツリーリンクをセットする。
;;;  内容の表示は、ツリーリンクの表示を使用する。

(:add-contents (lambda (self object)
 (let ((link
 ;; セットするツリーリンクを探す
 (card-system::find-object-from-title "Tree Link 1")))
   (and link
;; 自分自身(ケース)をルートにしてオブジェクトをノードにするリンクをセットする
(card-system::add-connection link self object) )))
 :after)

(:remove-contents (lambda (self object)
    (let ((link
    ;; セットするツリーリンクを探す
    (card-system::find-object-from-title "Tree Link 1")))
      (and link
   ;; リンクを削除する
   (card-system::remove-connection link self object) )))
 :after)

(:present-contents (lambda (self stream)
    (ignore self)
    (let ((link
    ;; ツリーリンクを探す
    (card-system::find-object-from-title "Tree Link 1")))
```

```
        (and link
    ;; 内容の表示を、ツリーリンクの表示を使用する。
    (card-system::present-contents link stream) )))
 )

;;; card-system manager

;;; :new-case :ater 新しくケースを作成したあとの処理。
;;;
;;;  デフォルトのスクリプトを与える。

(:new-case (lambda (self &rest options)
     (ignore options)
     (let ((default-script-string "(:add-contents
  (lambda (self object)
    ;; 使用しない
    (ignore self)
    ;; object の status を変更する
    (send object :set-status nil))
  :after)
\(:remove-contents
  (lambda (self object)
    ;; 使用しない
    (ignore self)
    ;;; どのケースにも属さないとき
    (unless (send object :cases)
      ;; object の status を変更する
      (send object :set-status t) ))
  :after)")
   (case (send self :get :create-object))) ; 作成されたケースの取り出し
       (and case ; ケースが存在するとき
    (send case :set-script-string default-script-string) )))
 :after)

;;; :call-card-window-menu カードウインドウのメニューを表示する。
;;;
;;;  特定(カードのタイプが :basic で、タイトルの先頭の二文字が "bc" のもの)
;;;  のオブジェクトについて表示するメニューの項目を変更する。
;;;
;;;  Next     = "sequence" リンク上のノードにあたる card を open する。
;;;  Previous = "sequence" リンク上のルートにあたる card を open する。

(:call-card-window-menu
```

```
  (lambda (self window)
    (let* ((type
     ;; カードのタイプを求める。
     (send (card-system::find-object-from-window window) :card-type))
   (menu-item-list
     ;; デフォルトのメニュー項目を求める。
     (cadr (assoc type (send self :card-window-operation-menu-item-alist))) )
   (object
     ;; オブジェクトを求める。
     (card-system::find-object-from-window window))
   (next
     ;; Next メニュー項目の定義。
     '(("Next"
:window-op
(lambda (window ignore ignore)
  (let ((card (card-system::find-object-from-window window))
next-card)
    (and card
 (let ((sequence-link
 ;; "sequence" リンクの検索。
 (card-system::find-object-from-title "sequence"
      :type :link)))
   (and sequence-link
(setf next-card
      ;; card の "sequence" リンク上のノードを求める。
      ;; card-system::search-tree は次のリストを返す。
      ;; (self root node-list)
      (first
(third
  (card-system::search-tree sequence-link card) ))))))
    (if next-card
(card-system::open-self next-card)
(beep) ))))))
   (previous
     ;; Previous メニュー項目の定義。
     '(("Previous"
:window-op
(lambda (window ignore ignore)
  (let ((card (card-system::find-object-from-window window))
previous-card)
    (and card
 (let ((sequence-link
 ;; "sequence" リンクの検索。
```

```
 (card-system::find-object-from-title "sequence"
      :type :link)))
   (and sequence-link
(setf previous-card
      ;; card の "sequence" リンク上のルートを求める。
      (second
(card-system::search-tree sequence-link card) )))))
    (if previous-card
(card-system::open-self previous-card)
(beep) )))))))
      ;; カードのタイプが :basic で、タイトルの先頭の二文字が "bc" のもの
      (when (and object
 (eq type :basic)
 (string= "bc" (send object :title) :end2 2) )
;; メニュー項目の追加。
(setf menu-item-list
      (append menu-item-list next previous) ))
      ;; メニューの表示。
      (let ((card-system::*card-window-operation-menu-item-list* menu-item-list))
(card-system::call-card-window-menu-internal self) )))
  )
```

第 17 章

付録 SDL 編集システム

第 18 章

SDL 編集システム 機能概要

18.1 SDL-Editor システムの構成

SDL-Editor システムには、カードシステムにおいて使用する SDL-Editor-card-system と、カードシステムとは切り離して単体で使用する SDL-Editor-Standalone とがある。

- SDL-Editor-card-system システム
- SDL-Editor-Standalone システム

18.2 SDL-Editor システムの使用方法

SDL-Editor システムには、カードシステムから起動する方法と単体で起動する方法がある。以下にそれぞれの起動方法について説明する。

- カードシステムのオブジェクトとして使用する方法
  Load Syatem コマンドで "SDL-Editor-Card-System" と入力する。
  ロードすることによりカードシステムのオブジェクトとして以下のものが追加され、カードシステムのオブジェクト選択メニューでそれぞれがセレクト可能になる。
  - SDL-GR
  - SDL-PR
  - SDL-Informal-PR
  - SDL-Case
- 単体で使用する方法
  Load System コマンドで "SDL-Editor-Standalone" と入力する。
  ロード後、<select>rectangle により SDL-GR 管理フレーム (Management-Window) をオープンし、SDL-Editor を起動する。また、<select>control-rectangle で新規に管理フレームを作成することができる。

【注意事項】　SDL-Editor を上記の使用方法で同時に起動することはできない。
なお、カードシステムの使用方法はカードシステムの『利用者マニュアル』を参照して下さい。

第 19 章

SDL 編集システム の起動法

19.1 機能概要

本プログラムは、通信システムの機能に関する仕様や動作を規定するために、CCITT により開発・勧告された SDL (Specification and Description Language) の図形エディタ (SDL-GR エディタ)、テキストエディタ (SDL-PR エディタ)、さらに簡略化した表現による SDL-GR 図変換機能を有するテキストエディタ (SDL-Informal-PR エディタ) をウィンドウ・ベースでサポートしています。これら 3 つのエディタは、カードシステム環境下でそれぞれ対応するカード (SDL-GR Card, SDL-PR Card, SDL-Informal-PR Card) として実現しており、それらのカードを一括管理するためのケース (SDl Case) も提供しています。

本支援プログラムは、カードシステム環境下で機能するシステム (SDL-Editor-card-system) であり、以下のカード、及びケースをサポートしています。

- SDL-GR Card
  SDL-GR 図 (状態遷移図) を作成するための図形エディタ (SDL-GR エディタ) を提供します。SDL-GR エディタは SDL-PR 表現に変換する機能を有しています。
- SDL-PR Card
  SDL-PR 表現のテキストを作成するためのエディタ (SDL-PR エディタ) を提供します。SDL-PR エディタは SDL-GR 図に変換する機能を有しています。
- SDL-Informal-PR Card
  簡略化した表現によって SDL テキストが作成できるエディタ (SDL-Informal-PR エディタ) を提供します。SDL-Informal-PR エディタは SDL-GR 図に変換する機能を有しています。
- SDL-Case
  上記の SDL 関連カードを複数個管理します。包含するカードに対して、対応関係をふまえながら表示する機能、論理規則に合致しているかのチェック機能、意味的な関連性を確認する機能を有しています。

また本支援プログラムは、カードシステムと関係なく SDL-GR エディタだけをサポートするシステム (SDL-Editor-standalone) も提供しています。

19.2 保存場所

SDL-Editor 関連のファイルは論理パス名としての card-system:sdl-editor; の下で管理されています。ATR 通信システム研究所では、その絶対パス名は

- `clm01:>local>binary>sdl-editor>genera-7-2>**>*.*.*`

- clm13:>local>binary>sdl-editor>genera-7-2>**>*.*.*

のどちらかが対応します。利用されるマシンのサーバマシンにより異なります。

以下で、個々のファイルのインストール先を述べます。

1. プログラム

   card-system:sdl-editor;*.lisp

   card-system:sdl-editor;*.bin

2. i-interface SDL データ

   card-system:sdl-editor;genera-7-2;sdl-data;*.lisp

19.3 起動方法

- SDL-Editor-card-system システムの起動

  1. load system Sdl-Editor-Card-System

     システム環境の設定がなされます。

  2. create card system manager Any :system type SDL-Editor-Card-System

     『SDL-Editor』用のマネージャを作成

  3. select + A

     『SDL-Editor』用のマネージャ画面の選択

  4. カードシステム・オブジェクトの生成

     - SDL-GR Card の作成

       Mouse-L でカード作成モードに入り、SDL-GR を選択します。その後、作成された SDL-GR をクリック（Mouse-L）することで SDL-GR エディタをオープンすることができます。

     - SDL-PR Card の作成

       Mouse-L でカード作成モードに入り、SDL-PR を選択します。その後、作成された SDL-PR をクリック（Mouse-L）することで SDL-PR エディタをオープンすることができます。

     - SDL-Informal-PR Card の作成

       Mouse-L でカード作成モードに入り、SDL-Informal-PR を選択します。その後、作成された SDL-Informal-PR をクリック（Mouse-L）することで SDL-Informal-PR エディタをオープンすることができます。

     - SDL Case の作成

       Mouse-M でケース作成モードに入り、SDL Case を選択します。その後、作成された SDL Case をクリック（Mouse-L）することで SDL Case をオープンすることができます。

     - データファイルの読み込み

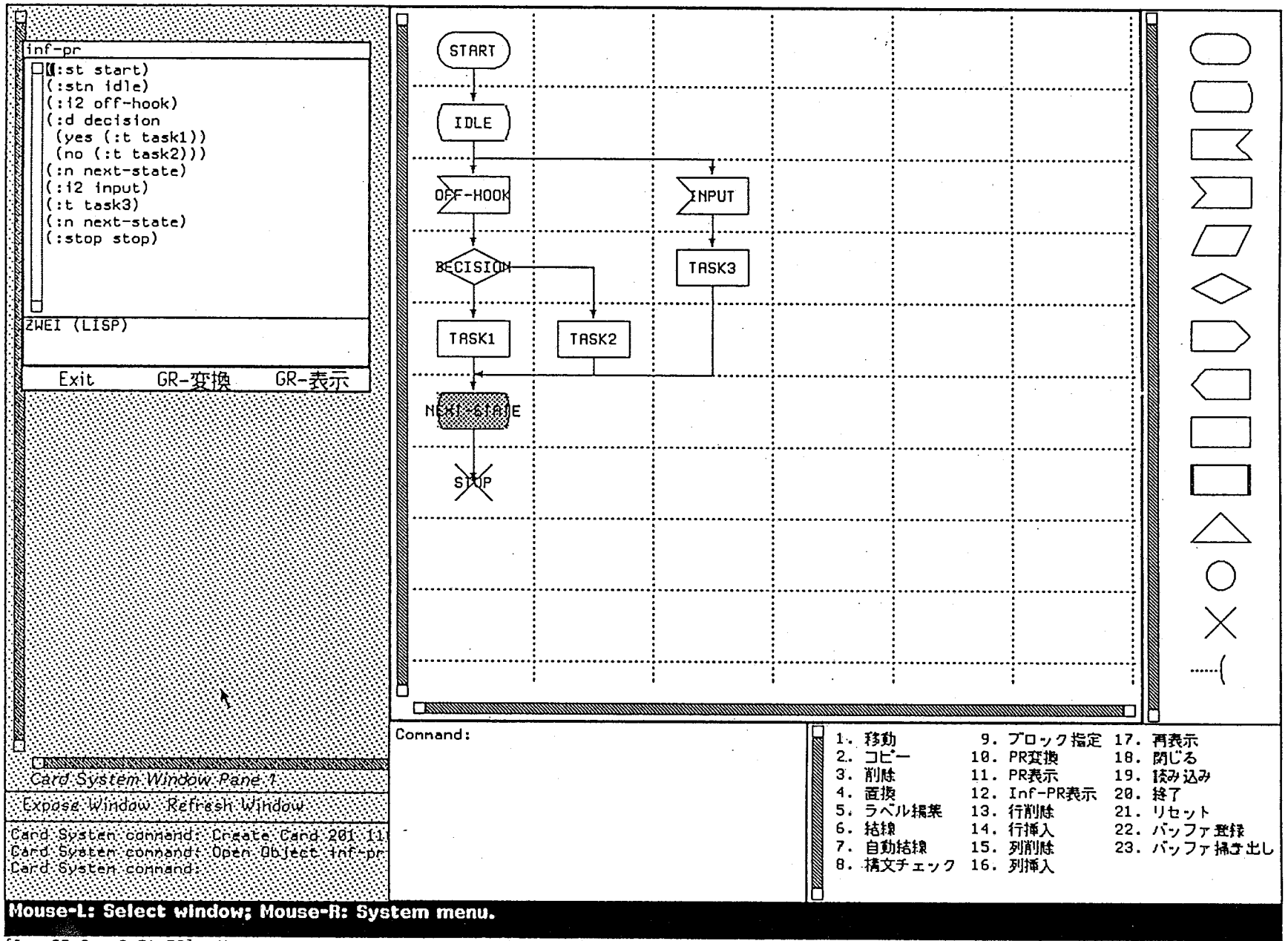
       カードシステム・データとして保存されているデータをロードし、再現します。

- SDL-Editor-standalone システムの起動

  1. load system Sdl-Editor-Standalone

     システム環境の設定がなされます。

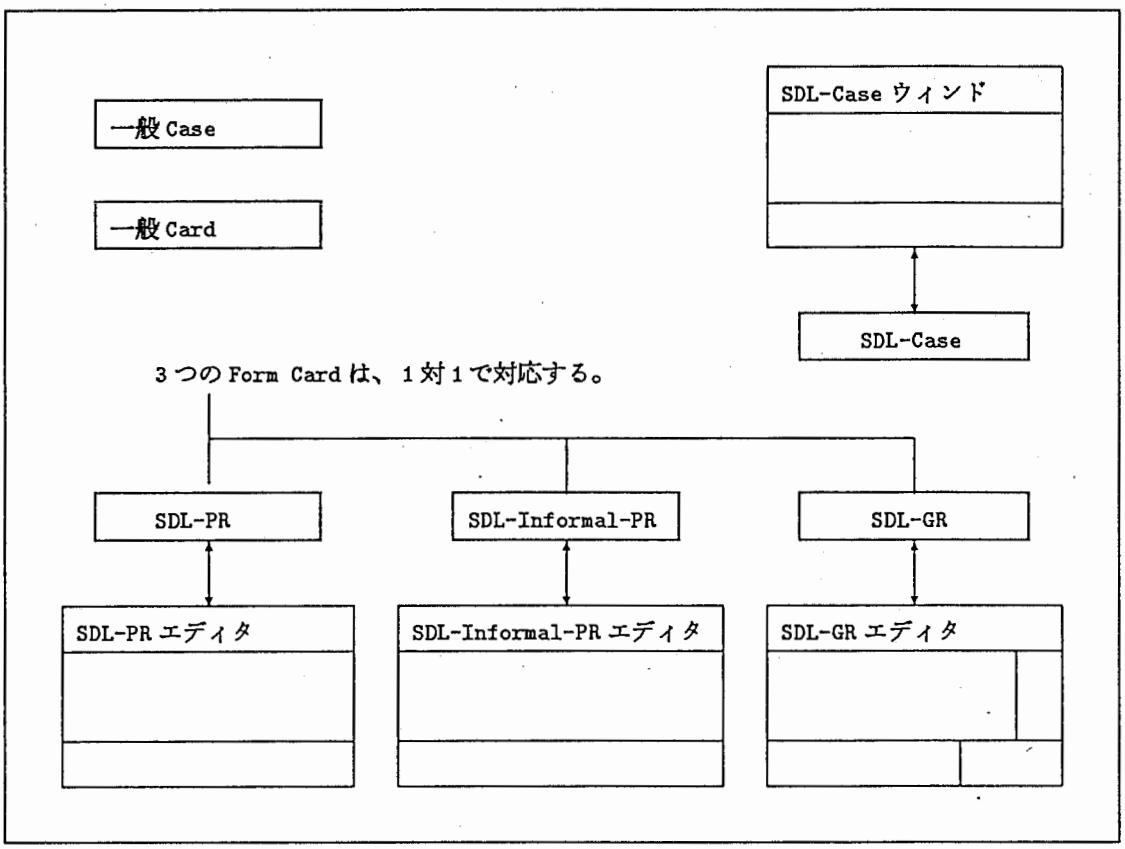
2. select + Rectangle
   管理フレーム画面の選択
3. Load Directory コマンド
   SDL 管理ディレクトリを指定することで、その下のデータファイルを読み込みます。
4. Load Sdl Text コマンド
   フレーム上に読み込まれたデータファイルをロードします。


inf-pr
(:st start)
(:stn idle)
(:i2 off-hook)
(:d decision
(yes (:t task1))
(no (:t task2)))
(:n next-state)
(:i2 input)
(:t task3)
(:n next-state)
(:stop stop)
ZWEI (LISP)
Exit
GR-変換
GR-表示
START
IDLE
OFF-HOOK
INPUT
DECISION
TASK3
TASK1
TASK2
NEXT-STATE
STOP
Card System Window Pane 1
Expose Window Refresh Window
Card System command: Create Card 201 11
Card System command: Open Object inf-pr
Card System command:
Command:
1. 移動
2. コピー
3. 削除
4. 置換
5. ラベル編集
6. 結線
7. 自動結線
8. 構文チェック
9. ブロック指定
10. PR変換
11. PR表示
12. Inf-PR表示
13. 行削除
14. 行挿入
15. 列削除
16. 列挿入
17. 再表示
18. 閉じる
19. 読み込み
20. 終了
21. リセット
22. バッファ登録
23. バッファ掃き出し
Mouse-L: Select window; Mouse-R: System menu.
[Tue 27 Sep 2:51:53] ikeno
CL-USER:
User Input

第 20 章

SDL-Editor システムの概要

20.1 SDL-Editor-card-system システムの概要



SDL-GR エディタは上図に示されるように、既存のカードシステムの上に 3 つの Form Card と 1 つの Form Case を定義することで実現されている。

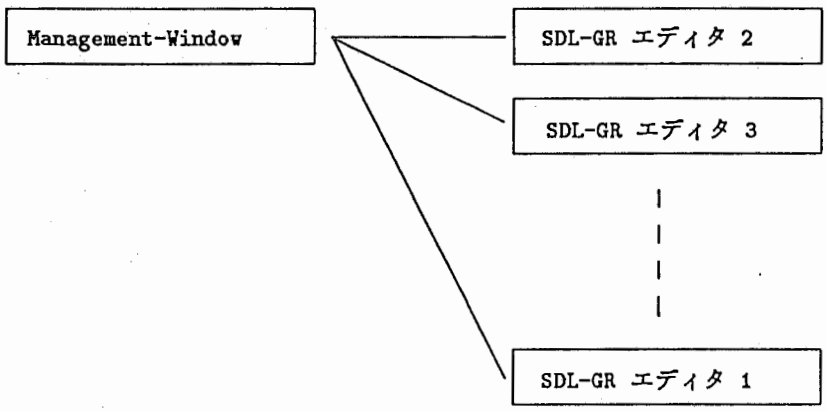
オブジェクト名	説　　明
SDL-Case	本ケースは、何枚か作成される部分SDLを格納するケースとして提供されている。本ケースはカードシステム内の他のケースと同様、他のケースに格納することも可能であり、また他のカード、ケースを格納することも可能である。
SDL-GR	本カードは、SDL-GRを編集するために提供されている。本カードはカードシステムの他のカードと同様、任意のケースへの格納やリンク付けの操作が可能である。
SDL-PR	本カードは、SDL-PRを編集するために提供されており、SDL-GRへの変換機能を持っている。本カードはカードシステムの他のカードと同様、任意のケースへの格納やリンク付けの操作が可能である。
SDL-Informal-PR	メモ書きレベルの状態遷移記述を本カード上に入力することにより、SDL-GRへ変換する機能を有している。本カードはカードシステムの他のカードと同様、任意のケースへの格納やリンク付けの操作が可能である。

20.2　SDL-Editor-Standaloneシステムの概要

SDL-Editor-Standaloneシステムは、SDL-Editor-card-systemにおけるSDL-GRエディタだけを切り離して利用する目的で作成したものである。

SDL-Editor-Standaloneシステムにおいては、SDL-PRエディタ、SDL-Informal-PRエディタについてはサポートされておらず、使用できない。

またSDL-Editor-card-systemシステムにおいては、カードシステムが各エディタを管理していたが、SDL-Editor-Standaloneシステムでは、複数のSDL-GRエディタを管理するためのManagement-Windowというフレームが存在する。Management-Windowの使用法については、『7．Management-Windowフレーム』において説明する。



第 21 章

SDL-GR

21.1 SDL-GR の作成

1. Create Card コマンドを実行
2. pop up 表示される Form card メニューより SDL-GR を選択
3. 属性の入力（タイトル等）

また、この他に SDL-PR エディタ、SDL-Informal-PR エディタ上において『GR 変換』コマンドを実行することによっても SDL-GR が作成される。
尚、この場合 SDL-GR が作成されると同時にそれに対応する SDL-GR エディタも作成される。

21.2 SDL-GR の Open,Close

- SDL-GR の Open
  （カードシステムのカードアイコン状態 → SDL-GR エディタ表示状態）
  カードシステムもつ他のカードを Open する操作と同様。
- SDL-GR の Close
  （SDL-GR エディタ表示状態 → カードシステムのカードアイコン状態）
  後に述べる SDL-GR エディタ内で『閉じる』コマンドを実行する。

21.3 SDL-GR 用のスクリプト

Keyword	引数	トリガー
:open-self	self	ウィンドウへのオープン指示
:close-self	self	ウィンドウへのクローズ指示
:move-self	self new-x new-y	アイコン、ウィンドウ移動指示
:present-self	self stream	アイコン表示
:delete-self	self	本カード削除指示
:add-case	self case	ケースへの登録指示
:add-link	self link	他のオブジェクトとのリンク指示
:remove-case	self case	ケースからの削除指示
:remove-link	self link	リンクからの削除指示
:auto-guide	self	アイコンメニュークリック時のマウス誘導
:auto-connection	self	自動結線指示
:syntax-check	self	構文チェック指示
:trans-pr	self	PR 変換指示

第 22 章

SDL-GR エディタ

22.1 SDL-GR エディタがもつ特徴

22.1.1 自動誘導・自動結線機能

SDL-GR エディタ画面が有する編集操作支援機能である。

- 自動誘導
  SDL-GR シンボルを作成する際にシステムがその挿入位置を想定し誘導するため、編集操作時のマウス移動が少なくて済む。
- 自動結線
  結線が行なわれていないシンボル間に対して、システムが想定して結線作業を行なう。

22.1.2 current-shape ノード連続作成機能

ノードの create スイッチの ON 指定（デフォルト）により、カレントの形状のノードを連続で作成できる。カレントな形状とは node-menu ペイン上で太枠表示された形状であり、マウスで指示することで自由に変更できる。

※ ノードの create スイッチ

node-menu ペインの最下段にスイッチアイコンが設定されてあり、マウス操作による ON/OFF 指定によりカレントな形状のノードを連続で作成する／しないをコントロールする。

22.1.3 マウス指示によるメッシュの出力 / 消去機能

command-menu ペイン内にメッシュ出力スイッチアイコンが設定されてあり、マウス操作による ON/OFF 指定により、メッシュの出力／消去をコントロールする。

22.1.4 Ａｒｃ編集機能

アークの通過経路を編集するために『Ａｒｃ編集コマンド』を用意しており、アークを編集することにより、希望する経路を通過するアークを得ることができる。

- 経路編集モード
  既定の起点・終点間に自由に通過経路を指定できる。
- 経路変更モード
  当該アークについてシステム側で考え得る通過経路を順次計算・出力し、希望の経路であれば選択できる。

以下に上記2つのモードの使用方法を説明する。

1. 経路編集モード

   - マウス入力
     - Node プレゼンテーション
       当該アークの行き先のノードである時に限り、最終通過ポイントからそのノードの入口に結線し終了する。他の場合はエラー。
     - Arc プレゼンテーション
       キャッチしたアークの行き先のノードが当該アークの行き先のノードと同一であるときに限り、最終通過ポイントから指定位置付近に垂直な矢（arrow）を下ろし終了する。他の場合はエラー。
     - 座標入力
       新規通過点となり、次の入力待ち状態となる。
   - キーボード入力
     - rubout キー
       最新（最後尾）の通過点を取り消す。
     - abort キー
       編集コマンド実行前のアークを出力し終了する。

2. 経路変更モード

   - キーボード入力
     - space キー
       起点・終点間の候補経路を順次出力する。
     - i キー
       起点を変更する。
     - o キー
       終点を変更する。
     - return キー
       現在の候補経路を確定採用する。

22.1.5 SDL-GR,PR,Informal-PR 間の変換機能 （※SDL-Editor-card-system システムの機能）

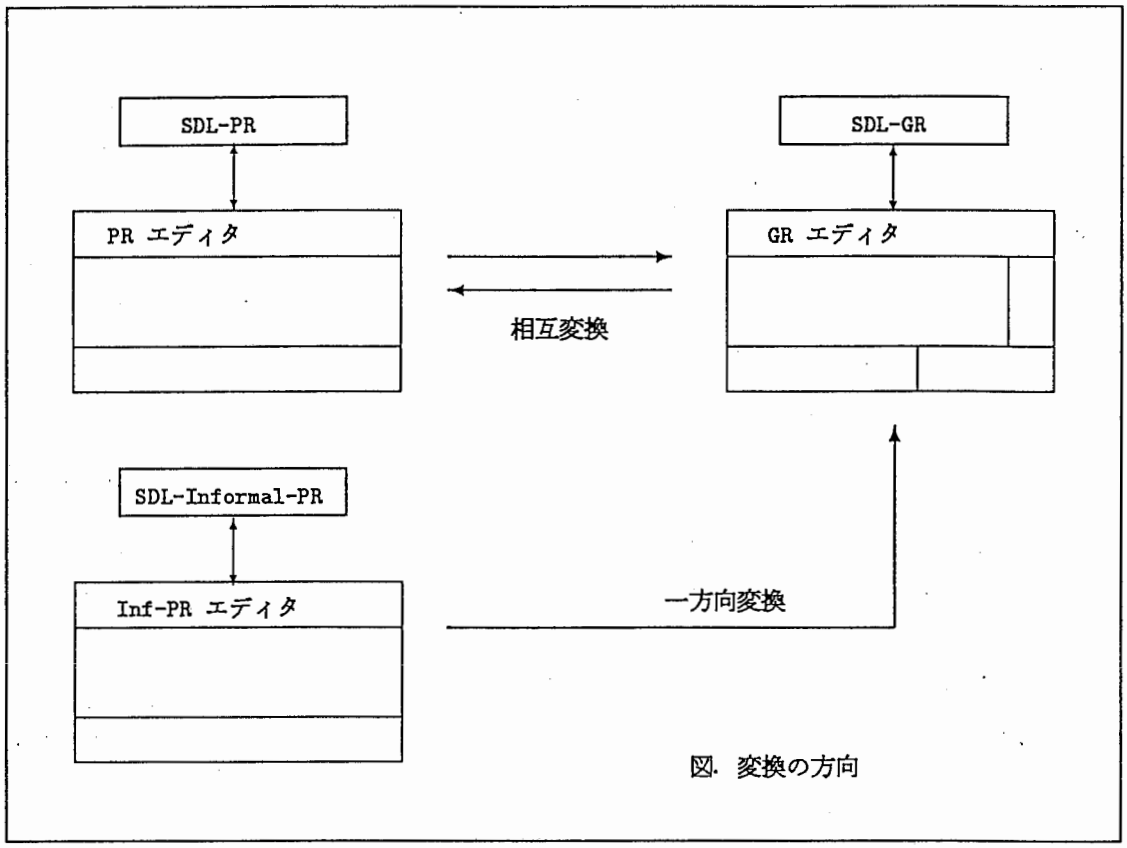


図. 変換の方向

上図で示される方向性でもって、SDL-GR・PR・Informal-PR 間の変換を行なうことができる。それら 3 種のカード間の対応は、常に 1 対 1 になるようシステムが管理しており、リンク付け等によるユーザ側での管理の必要はない。

22.1.6 検証支援機能

作成された状態遷移図に対して 3 つの視点（構文、論理、意味）からの検証支援機能を有する。
但し、構文チェック以外は SDL-Editor-card-system だけの機能である。

視点	説 明
構 文	シンボル間の連結ルールに対するチェック
論 理	仕様書の内部矛盾に対するチェック
意 味	原始要求に対する矛盾チェック支援

これらの検証機能の実現は、

構文検証は SDL-GR 画面が、
論理、及び意味検証支援は SDL-Case 画面が、

その役割を担っている。

22.1.7 構文チェック内容

次に示すノード連結ルールに基づいてチェックを行なう。

1. 部分 SDL は start ノードから始まっていなければならない。start ノードは他のノードから連結されない。
2. start ノードは次のノードへ連結される。state,stop,task,output,create-request,decision ノード
3. decision ノードは 2 本以上の decision アークを持たなければならない。
4. decision アークは名前を持つアークであり、state,stop,task,output,create-request,decision ノードに連結される。
5. state ノードは 1 つ以上の input ノードまたは save ノードへ連結されなければならない。
6. input ノードは唯一の state ノードから連結されなければならない。
7. input ノードは次のノードへ連結される。state,stop,task,output,create-request,decision ノード
8. stop ノードから連結されるノードはない。
9. 部分 SDL は 2 つ以上の stop ノードを持たない。
10. 全てのシンボルはつながっていなければならない。

※ 構文チェック場面においては、input,output ノードについての方向（input1/2,output1/2）の区別はない。

22.2 SDL-GR エディタの作成方法

既に説明したように SDL-Editor システムには 2 種のシステムがあり、各システムで独自の SDL-GR エディタが作成される。

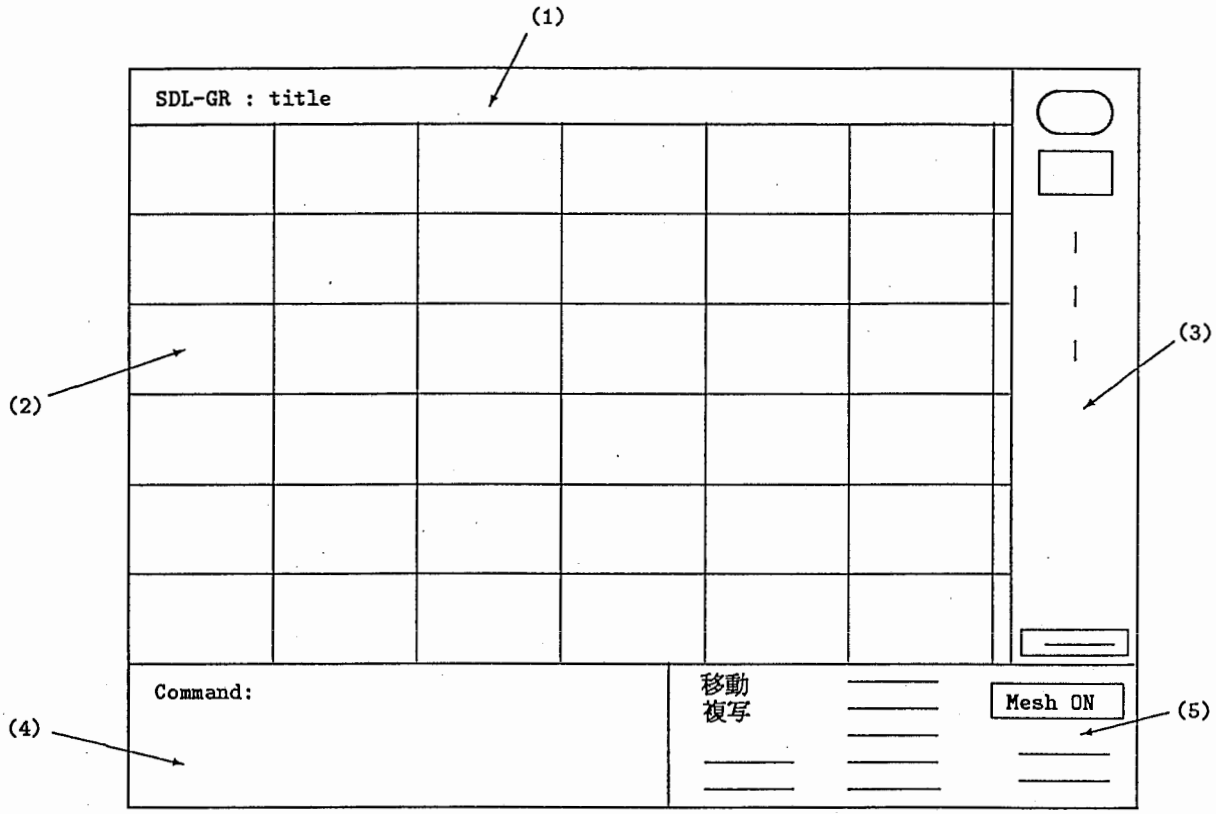
- SDL-Editor-card-system システムでの作成

  カードシステム内で SDL-GR を作成し、その後作成された SDL-GR をオープンする。
  または SDL-PR エディタ、SDL-Informal-PR エディタ画面上で『GR 変換』コマンドを実行する。
  この場合、SDL-GR エディタが作成されると同時に SDL-GR も作成される。

- SDL-Editor-standalone システムでの作成

  別紙参照。

22.3 SDL-GR エディタの画面構成



1. title ペイン
   タイトルを出力する。

2. graph ペイン
   SDL 図を出力する。

3. node-menu ペイン
   出力ノードのアイコンメニュー

4. commands ペイン
   interactor

5. command-menu ペイン
   編集コマンドのメニュー

22.4 SDL-GR エディタがもつアイコンメニュー

SDL-GR エディタは次に挙げるアイコンをメニューとしてもつ。

これらの任意のシンボルを #\mouse-1 で選択し、表示位置を SDL-GR 編集画面上（graph ペイン）で指示（#\mouse-1）することで入力が行なわれる（自動誘導機能）。但し、指示された位置に既に別のノードが存在する場合には挿入と見なして処理している。

- 自動誘導機能メニューアイコンをマウス選択（#\mouse-1）した際に作成位置を次の条件に従って想定し、マウスを自動的に誘導する。

  1. input 及び comment ノード以外はカレントシンボルの直下の枠。
  2. input ノードは第一 state ノードの 1 段下の空の枠。
  3. comment ノードはカレントシンボルの右、または左隣の枠。
  4. 上記の条件に合致する枠に既に別のノードが存在する場合は、その下方向の空いている枠。

【注意事項】
カレントシンボルとは、コマンド実行対象シンボル（後述）を指す。

22.5 SDL-GR エディタがもつコマンドメニュー

SDL-GR エディタは次に挙げるコマンドをメニューとしてもつ。

- 共通コマンド（両方のシステムでサポート）

1) 移動	2) 複写	3) 削除	4) 形状置換
5) ラベル編集	6) 結線	7) 自動結線	8) ノード検索
9) ラベル置換	10) アーク編集	11) 構文チェック	12) タイトル編集
13) バッファ登録	14) バッファ掃き出し	15) ファイル読み込み	16) ファイル書き込み
17) 閉じる			

- SDL-Editor-card-system システム専用コマンド

18)PR 変換　19)PR 表示　20)Inf-PR 表示

【注意事項】上記コマンドの内、1) ～ 7),10),13) についてはカレントシンボル（下記）指定した後に起動をかける。

22.5.1 カレントシンボル (編集操作対象)

カレントシンボル（カレントオブジェクトとも表現している）とは、画面上で特殊表示（ノードの場合網掛け表示、アークの場合太線表示）されている、直前に作成あるいは編集されたシンボル（ノードまたはアーク）であり、編集操作の対象となっている。

SDL-GR エディタ内においては、一部を除くコマンドについてはすべてカレントシンボルに対して働くようになっている。

- カレントシンボル設定方法

  - 一品指定
    ノード、またはアークに対してマウス指示（#\mouse-1）
  - ブロック（複数）指定
    SDL-GR エディタ画面上（graph ペイン）でマウスが何もセンスしていない状態でマウス指示（#\mouse-1*2）、またはコマンド『ブロック指定』により、ブロック指定モードに入る。ブロック指定はマウスによるウィンドウの edges 指定のような操作で行なう。（破線枠により誘導）

22.5.2 コマンド説明

- 移動
  移動先を指定する。挿入移動は、移動先としてノードの存在する枠を指定することでその上に移動される。
  操作対象は一品、ブロック共に可。

- 複写
  コピー先を指定する。操作上の手順は『移動』に同じ。
  操作対象は一品、ブロック共に可。

- 削除
  操作対象は一品、ブロック共に可。

- 形状置換
  ノードシンボルの形状を置換する。文字情報、アーク情報等はすべて有効。
  操作対象は一品のみ可。

- ラベル編集
  ノード、またはアークのラベルを設定する。
  操作対象は一品のみ可。

- 結線
  2ノード間にアークを引く。
  操作対象は一品のみ可。

- 自動結線
  アーク情報を持たないノード間に結線する。操作対象がない場合、または操作対象が単一の場合は全ノード間に対して処理する。

- ノード検索
  指定のラベルをもつノードを検索する。

- ラベル置換
  指定の文字列をラベルにもつノードを別の文字列に置換する。

- アーク編集
  アークの通過経路を編集する。（Ａｒｃ編集機能参照）

- 構文チェック
  Process Graph でのノード連結ルールについて検査を行なう。操作対象がない場合、または操作対象が単一の場合は、全ノードを対象にチェックを行なう。
  連結ルールについては構文チェック内容参照。

- タイトル編集
  SDL-GR エディタのタイトルを設定する。

- バッファ登録
  現在の操作対象の内容を専用変数に確保する。

- バッファ掃き出し
  専用変数にある内容を位置を指定し、出力する。

- ファイル読み込み
  ファイル名を入力することで、SDL テキストファイルをロードする。

- ファイル書き込み
  現在の SDL 図情報をファイルにセーブする。

- 閉じる
  SDL-GR エディタをクローズする。

22.5.3 SDL オペレーションメニュー

現在の操作対象シンボルに対するコマンドの一覧と以下に説明する SDL ウィンドウメニュー、SDL ビハインドメニューを 1 つのメニューとしてまとめて表示する。

22.5.4 SDL ウィンドウメニュー

SDL-GR エディタ上でのウィンドウ操作に関するコマンドの一覧を表示する。

- 再表示
  画面の情報をすべて消去して書き直す。

- リフレッシュ
  画面をリフレッシュする。

- リセット
  画面の情報をすべて消去してクリアする。

- ブロック指定
  破線枠で誘導し複数ノードをカレントシンボルに指定する。

- 行挿入
  指定行上に指定行数分だけ空行を挿入する。

- 行削除
  指定行上から指定行数分だけ行を削除する。

- 列挿入
  指定列上に指定列数分だけ空列を挿入する。

- 列削除
  指定列上から指定列数分だけ列を削除する。

22.5.5 SDL ビハインドメニュー

その他の操作コマンドメニュー

- ノードサイズ変更
  ノードの基準となる大きさを一品別、シンボル別、全体に対して変更する。

- メッシュサイズ変更
  画面上に出力されるメッシュ枠のサイズを変更する。

- カレント表示消去
  操作対象シンボルについて特殊表示（ノードの場合網掛け表示、アークの場合太線表示）されているものを元に戻す。

- 次ページ表示
  次ページを表示する。

- 前ページ表示
  前ページを表示する。

- 右ページ表示
  右ページを表示する。

- 左ページ表示
  左ページを表示する。

22.6 SDL-GR エディタ上のマウスジェスチャ

SDL-GR エディタ上では次にあげるマウス操作をサポートしている。

- ノードシンボルをセンスしている時

left	カレントシンボル指定
middle	ラベル編集
shift-left	結線
shift-middle	移動
control-middle	削除
meta-left	複写
hyper-left	構文エラー表示
hyper-middle	形状置換

- アークシンボルをセンスしている時

left	カレントシンボル指定
middle	ラベル編集
control-middle	削除
meta-left	アーク編集

- 何もセンスしていない時

left	カレントな形状のノードを作成
shift-left	ブロック指定
meta-left	SDL Window メニューの出力
meta-middle	SDL Behind メニューの出力
meta-right	SDL Operation メニューの出力

第 23 章

SDL-Case

23.1 SDL-Case の作成方法

SDl-Case は次の手順で作成できる。

1. Create Case コマンドの選択
2. pop-up 表示される form case メニューより SDL-Case の選択
3. 属性の入力（タイトル等）

23.2 SDL-Case の open,close

- Open （アイコン状態 → 内容表示状態）
  カードシステムがもつ他のケースをオープンする操作と同様。
- Close （内容表示状態 → アイコン状態）
  コマンドメニュー内の『閉じる』を選択する。

23.3 SDL-Case がもつコマンドメニュー (SDL 専用のコマンド)

以下のコマンドは SDL-Case ウィンドウ上で、pop up 表示される（何もセンスしていない状態で #\mouse-r による）メニューから選択することにより起動できる。

- 論理チェック
  SDL-Case 内に格納されている SDL-GR を対象に（実際にはその SDL-GR に対応する SDL-GR エディタ上の SDL 図を対象としている）、次に示す論理規則に基づいてチェックを行ない、規則に反するものについては SDL-GR オブジェクトのボックスを反転表示する。

  チェック内容

  1. save ノードがある場合、次の状態でその信号名と同じ input ノード、または save ノードが必要である。
  2. 各状態に定義されている input ノードと save ノードの信号名はすべて異なる必要がある。
  3. output ノードと同一の信号名をもつ input ノードが必要である。

※ 信号名とはラベル名のことである。また input,output ノードについては、方向 (1/2) の区別はない。

- 意味誤り確認

  SDL-Case 内に格納されている SDL-GR を対象に出力信号（output ノード）、入力信号（input ノード）の信号名でマッチングをかけ、関連する SDL-GR オブジェクトを対応付けて表形式で出力する。

23.4 SDL-Case 用スクリプト

Keyword	引数	トリガー
:open-self	self	ウィンドウへのオープン指示
:close-self	self	ウィンドウへのクローズ指示
:move-self	self new-x new-y	アイコン、ウィンドウ移動指示
:present-self	self stream	アイコン表示
:add-contents	self object	オブジェクトの本ケースへの登録指示
:remove-contents	self object	本ケースよりオブジェクトの削除処理
:present-contents	self stream	本ケース内容の表示指示
:delete-self	self	本ケースの削除指示
:add-case	self case	他のケースへの登録指示
:add-link	self link	他のオブジェクトとのリンク指示
:remove-case	self case	他のケースからの削除指示
:remove-link	self link	リンクからの削除指示
:logical-check	self	論理チェック指示
:semantic-check	self	意味チェック指示

第 24 章

SDL-Informal-PR

24.1 SDL-Informal-PR の作成方法

SDL-Informal-PR は次の手順で作成できる。

1. Create Card コマンドの選択
2. pop up 表示される Form card メニューより SDL-Informal-PR の選択
3. 属性の入力（タイトル等）

24.2 SDL-Informal-PR の open,close

- Open （アイコン状態 → 内容表示状態）
  カードシステムがもつ他のカードをオープンする操作と同様。
- Close （内容表示状態 → アイコン状態）
  コマンドメニュー内の『閉じる』を選択する。

24.3 SDL-Informal-PR の入力方法

本カードの編集機能は、Symbolics マシンが提供している zwei:standalone-editor のそれに従い入力を行なう。入力フォーマットは次に示す S 式表現である。

表 24.1: SDL-Informal-PR 用シンボル略号表

略号	正式名称	略号	正式名称
:st	Start	:t	Task
:stn	State	:a	Alternative
:i1	Input （右向き）	:j	Join or Connector
:i2	Input （左向き）	:n	Nextstate
:s	Save	:stop	Stop
:d	Decision		
:o1	Output （右向き）		
:o2	Output （左向き）		

※ 略号に合わないものが入力されている場合、SDL 図への変換時には特にエラーとしては扱わず、すべて task ノードとして扱う。

24.4 SDL-Informal-PR がもつコマンドメニュー

閉じる　ウィンドウを閉じ、アイコン表示状態に戻る。
m-x コマンド（Close）でも同様。

GR 変換　SDL 図への変換処理を行ない、対応する SDL-GR エディタをオープンし出力する。
m-x コマンド（GR-Compile）でも同様。

GR 表示　対応する SDL-GR エディタをオープンする。
m-x コマンド（GR-Open）でも同様。

24.5 SDL 図への変換例

```
((:st "start")
 (:stn "idle")
 (:i2 "off-hook")
 (:d "decision"
     ("yes" (:t "task1"))
     ("no" (:t "task2")))
 (:n "nextstate")
 (:i2 "input")
 (:t "task3")
 (:n "nextstate")
 (:stop "stop"))
```

24.6 SDL-Informal-PR 用スクリプト

Keyword	引数	トリガー
:open-self	self	ウィンドウへのオープン指示
:close-self	self	ウィンドウへのクローズ指示
:move-self	self new-x new-y	アイコン、ウィンドウ移動指示
:present-self	self stream	アイコン表示
:delete-self	self	本カードの削除指示
:add-case	self case	ケースへの登録指示
:add-link	self link	他のオブジェクトとのリンク指示
:remove-case	self case	ケースからの削除指示
:remove-link	self link	リンクからの削除指示
:trans-GR	self	SDL 図変換指示

第 25 章

SDL-PR

25.1 SDL-PR の作成方法

SDL-PR は次の手順で作成できる。

1. Create Card コマンドの選択
2. pop up 表示される Form card メニューより SDL-PR の選択
3. 属性の入力（タイトル等）

※ この他 SDL-GR エディタ上での『PR 変換』コマンドによっても SDL-PR を作成することができる。但しこの場合、対応する SDL-PR エディタが既に存在するならば、それはクリアされるので注意が必要。

25.2 SDL-PR の open,close

- Open （アイコン状態 → 内容表示状態）
  カードシステムがもつ他のカードをオープンする操作と同様。
- Close （内容表示状態 → アイコン状態）
  コマンドメニュー内の『閉じる』を選択する。

25.3 SDL-PR がもつコマンドメニュー

閉じる	ウィンドウを閉じ、アイコン表示状態に戻る。 m-x コマンド（Close）でも同様。
GR 変換	SDL 図への変換処理を行ない、対応する SDL-GR エディタをオープンし出力する。 m-x コマンド（GR-Compile）でも同様。：未開発
GR 表示	対応する SDL-GR エディタをオープンする。 m-x コマンド（GR-Open）でも同様。：未開発

25.4 SDL 図への変換例

```
process pr1;
  state st01;
    input sn1;
      decision d1;
      (no): output sn3;
      (yes): decision d2;
             (no): task tsk1;
             (yes): task tsk2;
             enddecision;
      output sn2 to pr2;
      enddecision;
  nextstate st01;
endprocess pr1;
```

25.5 SDL-PR 用スクリプト

Keyword	引数	トリガー
:open-self	self	ウィンドウへのオープン指示
:close-self	self	ウィンドウへのクローズ指示
:move-self	self new-x new-y	アイコン、ウィンドウ移動指示
:present-self	self stream	アイコン表示
:delete-self	self	本カードの削除指示
:add-case	self case	ケースへの登録指示
:add-link	self link	他のオブジェクトとのリンク指示
:remove-case	self case	ケースからの削除指示
:remove-link	self link	リンクからの削除指示
:trans-GR	self	SDL 図変換指示

第 26 章

Management-Window フレーム

Management-Window フレームとは、SDL-Editor-Standalone システム専用の SDL-GR エディタウィンドウを管理するものであり、管理下の SDL 図についてファイル名、タイトル名等の情報を表形式で出力している。

26.1 ファイル管理表の項目説明

- delete-p
  削除マークの有無。削除マーク付きファイルに付いては "D" を出力。
- filename
  SDL テキストファイル名。
- sdl-title
  SDL テキストファイルに設定されている SDL 図のタイトル。
- open-p
  対応する SDL-GR エディタウィンドウが既に作成されているならば "*" を出力。

26.2 command-menu の項目

- Create Text
  新規に SDL テキストファイルを作成する。
- Delete Text
  ファイル管理表上の SDL テキストファイルを削除する。
- Load Directory
  SDL テキストファイルを管理しているディレクトリを指定することにより、そのディレクトリ下の最新ファイルを読み込みファイル管理表に出力する。
- Load Text
  指定の SDL テキストファイルをロードし、SDL-GR エディタウィンドウをオープンする。
- Redisplay
  ファイル管理表上の内容を最新の情報に基づいて書き直す。

26.3 マウス起動コマンド (translator)

- load-text-file
  ファイル管理表上で任意の行を選択することにより、SDL テキストファイルをロードし、対応する SDL-GR エディタウィンドウを open する。

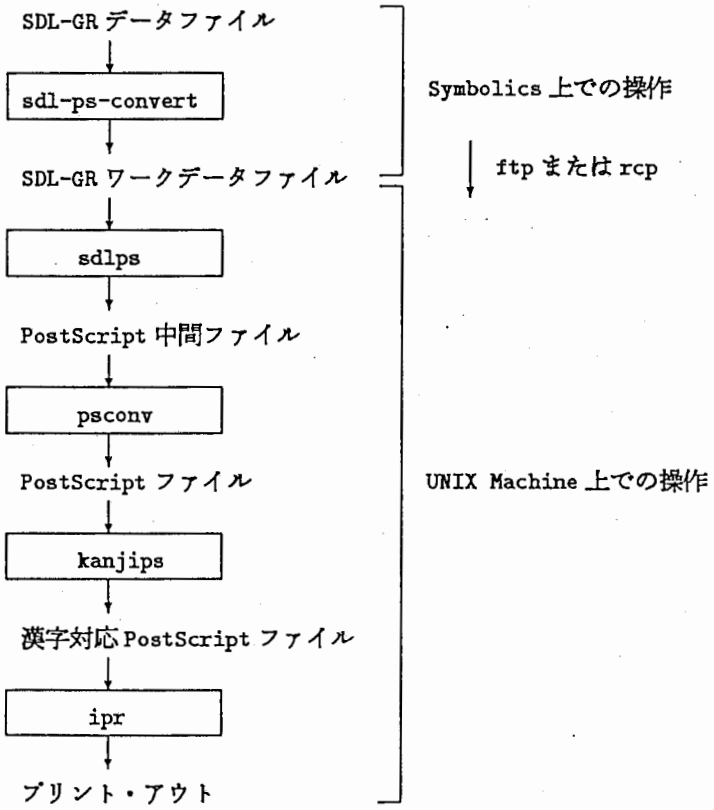
【注意事項】

- 指定のディレクトリには SDL テキストファイル以外のファイルがないことを前提とする。
- 最新で削除マークのないファイルを処理対象としているので、削除マーク付き、および旧バージョンのファイルはファイル管理表上に読み込まない。

第 27 章

付録 SDL データの Postscript 変換システムの概要

本システムは、Symbolics 上で作成した SDL-GR データを用いて、UNIX マシン上で
PostScript 形式のファイルに変換するものである。
SDL-GR データから PostScript 形式のファイルに変換するまでの過程を以下に示す。



第 28 章

プログラム概要

本システムのプログラムは次の 3 つから構成されている。

- sdl-ps-convert
  SDL-GR データを UNIX で読み取り易い形式に変換する。
  - リスト構造のデータの分解
  - 漢字コードの変換 (Symbolics 漢字コード → JIS 漢字コード)
- sdlps
  SDL-GR データを PostScript 中間ファイルの形式に変換する。変換時の機能として以下のものがある。
  - 用紙サイズの選択 (A4 または B4)
  - 描画縮尺の指定
  - メッシュの描画の指定
  - 描画方向の指定 (縦、横)
  - フォントの切り替え (jisj フォント、dmj フォント)

  (注) 半角文字列は、自動的に全角文字列に変換される。
- psconv
  PostScript 中間ファイルを PostScript ファイルに変換する。

第 29 章

ファイル説明

29.1 SDL-GR データファイル

- 環境 (先頭の S 式)

キーワード	説明	データタイプ
:title	SDL 図のタイトル	string
:mesh-height	メッシュ高さ	int
:mesh-width	メッシュ幅	int
:create-mesh-height	ノード作成時のデフォルト高さ	int

- ノード

キーワード	説明	データタイプ
:node	オジェクトの種類	
:unique	ノード管理インデックス	int
:xpos	メッシュ座標 (X)	int
:ypos	メッシュ座標 (Y)	int
:label	ノードラベル	string
:shape	ノードの形状	keyword-symbol
:size	ノードの大きさ	int
:in-direction	入力アークが入る方向	keyword-symbol
:out-direction	出力アークが出ていく方向	keyword-symbol
:select-flg		

(補足)

- shape の種別

:start	:save	:task	:stop	:procedure-input2
:state	:dicision	:pocedure-call	:procedure-start	:procedure-output1
:input1	:output1	:alternative	:return	:procedure-output2
:input2	:output2	:conector	:procedure-input1	:comment

- direction の種別

:top :under :left :right

- アーク

キーワード	説明	データタイプ
:arc	オブジェクトの種類	
:from-node-unique	起点ノードの管理インデックス	int
:to-node-unique	終点ノードの管理インデックス	int
:label	アークラベルと表示位置	list
:dependent-arc		
:independent-arcs		
:draw-point	アークの全経路と矢印部分の経路	list
:editted		
:arc-type	アーク種別 (:ordinary :comment)	keyword-symbol
:select-flg		

(補足)

- label
  (string (X 座標 Y 座標))
- draw-point
  ((X 座標 Y 座標 X 座標 Y 座標) - (X 座標 Y 座標 X 座標 Y 座標))

29.2 SDL-GR ワークファイル

- 環境部

:environment	環境データの記述開始
:title	タイトル
"ＳＤＬデータ"	
:mesh-height	メッシュ高さ
30	
:mesh-width	メッシュ幅
20	
:end-environment	環境データの記述終了

- ノード部

:node	ノードデータの記述開始
:unique	ノードユニークＩＤ
1	
:xpos	ノードのメッシュ位置 (X 座標)
2	
:ypos	ノードのメッシュ位置 (Y 座標)
4	
:label	ノードのラベル
"ノードラベル"	
:shape	ノードの形状
:task	
:size	ノードサイズ
20	
:in-direction	入力アークの方向
:top	
:out-direction	出力アークの方向
:under	
:end-node	ノードデータの記述終了

- アーク部

:arc	アークデータの記述開始
:form-node-unique	開始ノードのユニーク ID
1	
:to-node-unique	終了ノードのユニーク ID
2	
:label	アークラベル
"yes"	
:label-position	アークラベルの表示位置
210 300	
:draw-points	アークの全経路
3	構成点数
200 300 100 200 300 300	構成点座標
:arrow-points	矢線の構成点座標
100 200 300 300	
:arc-type	アークの種別
:ordinary	
:end-arc	アークデータの記述終了

- コメント部

  第1カラムが ";" で始まるレコードをコメントとみなす。

29.3 Post-Script 中間ファイル

29.3.1 出力プリミティブ

- 線分

  :line

  開始・終了点の座標を指定することにより線分を描画する。

  (100 100) (100 200)

  :lines

  折れ線の構成点数とその座標を指定することにより折れ線を描画する。

  4 (100 100) (100 200) (200 200) (200 100)

  :box

  長方形の対角の頂点の座標を指定することにより長方形を描画する。

  (100 100) (200 200)

  :polygon

  頂点の数とその座標を指定することにより多角形を描画する。

  4 (100 100) (100 200) (200 200) (200 100)

  :arrow

  矢線の開始・終了点の座標を指定することにより矢線を描画する。(100 100) (100 200)

- 円

  :circle

  半径と中心の座標を指定することにより円を描画する。

  10 (100 100)

- 円弧

  :arc

  半径と中心の座標、開始・終了角を指定することにより円弧を描画する。

  10 (100 100) 0 90

- 文字列

  :string

  描画位置と文字列より文字を描画する。

  (100 100) "ABC"

29.3.2 その他

- 属性

  以下の属性指定が行われると属性変更が行われるまで指定された属性は引き継がれる。

  :fill

  長方形、多角形、円、円弧の中塗りの指定である。

  デフォルトは off である。

  on　または　off

  :line-width

  線分を描画するときの線幅の指定である。

  デフォルトは 2 である。

  2

  :font

  文字列を描画するときのフォント指定である。

  デフォルトは helvetica である。

  helvetica

  :font-size

  文字の高さを指定する。

  デフォルトは 10 である。

  10

- 制御

  :end-page

  ページ切り替えの指定である。

  :end-of-file

  データの終わりを示す。

  :special

  直接 PostScript の構文を記述することができる。

  :file

  別の PostScript 中間ファイルを読み込んで処理を行う。

第 30 章

SDL – Postscript 変換システムの起動法

30.1 機能概要

SDL システムの"save" コマンドで作成されたデータファイル (SDL-GR 図) を PostScript 対応のファイルに変換し、Imagen プリンタへの出力を可能にする。

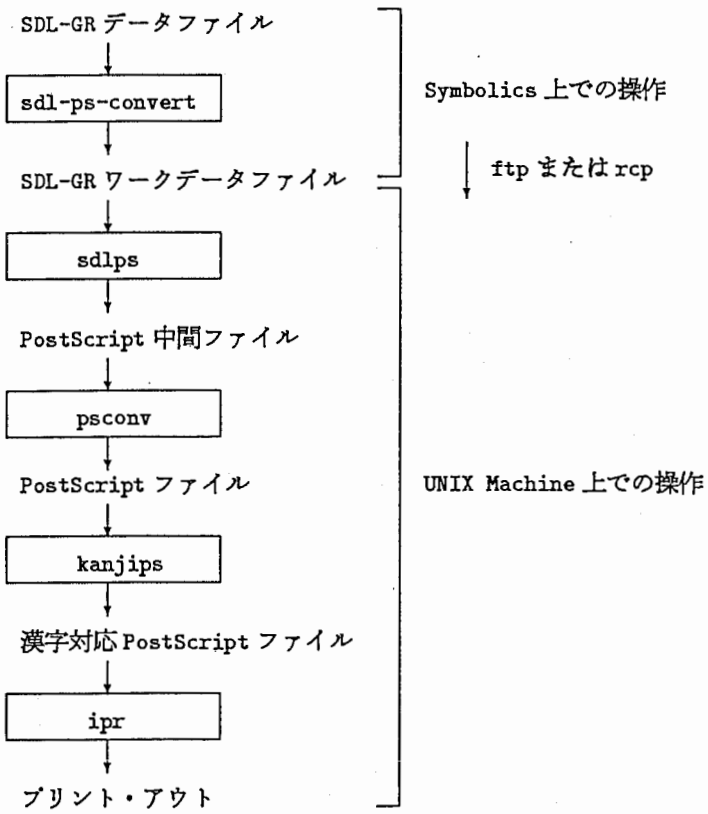
30.2 保存場所

1. Symbolics : `CLM01:>local>binary>sdl2ps>sdl-ps-convert.lisp`
2. UNIX : /utility/sdl2ps/sdlps.c, psconv.c, Makefile

30.3 起動方法

作業フロー

本システムを用いて、SDL-GR データを PostScript 形式のファイルに変換し、
プリンタに出力する際の作業フローを以下に示す。



30.3.1 sdl-ps-convert

SDL-GR データファイルを UNIX 側で読み取り易い形に変換する。

- Symbolics 漢字コードを JIS 漢字コードに変換する。
- リスト構造のデータを分解する。

<関数使用方法>

(sdl-ps-convert SDL-GR データファイルのパス名)
Enter output-filename: SDL-GR データワークファイル名

<関数使用例>

```
Command: (sdl-ps-convert #P"CLM13:cosmossdl-datajt-q931.lisp")
Enter output-filename: sdl-work.lisp
Command:
```

30.3.2 sdlps

SDL-GR データワークファイルを PostScript 中間ファイルに変換する。
コマンドの変換機能は以下の通りである。

- メッシュの描画の選択
- 用紙サイズの選択 (A4, B4)
- 描画縮尺の指定

- 描画方向の指定（縦, 横）
- フォントの変更

＜コマンド使用方法＞

```
$ sdlps [-m] [-p 用紙サイズ] [-s 描画縮尺] [-h] [-f フォント名]
  SDL-GR データワークファイル名 > PostScript 中間ファイル名
```

＜コマンドオプション説明＞

-m	メッシュの描画指定 省略時は描画しない。
-p ＜用紙＞	用紙の指定（A4 4） 省略時は A4 である。
-s ＜縮尺＞	縮尺（0.0 ＜ scale ＜ 1.0） 省略時は 1.0 である。 （目安： A4 横 - 0.5 ,B4 横 - 1.0 程度）
-f ＜フォント＞	フォントの指定 指定できるフォントには以下のものがある。 dmj6,dmj7,dmj8,dmj9,dmj10,dmj12,dmj17,dmj20 jisj24,jisj36,jisj48,jisj72 省略時は dmj8 である。
-h	用紙の方向を横にする。 省略時は縦である。

＜コマンド使用例＞

```
$ sdlps -m -p A4 -s 0.5 -f dmj9 -h sdl.data  postscript.data
$
```

＜注意事項＞

当コマンドを実行すると、work,title_work,font_work というファイルが自動的に作成されますが、これは、次のステップで使用するファイルのためそのまま置いておくこと。

30.3.3 psconv

PostScript 中間ファイルを PostScript 形式のファイルに変換し、標準出力に表示する。

＜コマンド使用方法＞

```
$ psconv PostScript 中間ファイル名  PostScript ファイル名
```

＜コマンド使用例＞

```
$ psconv postscript.data > postscript.src
```

または、

```
$ sdlps sdl.data | psconv > postscript.src
```

<注意事項>

当コマンドでは、ファイル名が指示されなかった場合は、標準入力からデータを読み込むように設計してある。そのため、コマン使用例の後者の形の実行方法も行うことができる。

30.4 kanjips および ipr

漢字対応でない PostScript プリンタに漢字データを出力する。

<実行方法>

```
$ kanjips PostScript ファイル名 | ipr -L Ultrascript -Pimagen
```

第 31 章

付録 Sequence chart 編集システム

第 32 章

Sequence chart 機能概要

本システムは、信号シーケンス図を作成するための編集支援ツールである。
主な機能として、

1. 入出力主体（ノード）、信号（シグナル）の作成・編集機能
2. SDL-GR 図への変換機能
3. 信号シーケンス図のファイルへの登録・呼び出し機能

をもつ。

第 33 章

Sequence chart の起動法

33.1 機能概要

本プログラムは、通信プロトコルを表現するための信号シーケンス図編集支援プログラムである。

本支援プログラムはカードシステム環境下で Signal-Sequence システムとして実現されており、SDL-GR 図への変換機能を有した信号シーケンス・エディタをサポートしています。

信号シーケンス・エディタは、カードシステム上ではカードオブジェクト（Signal-Sequence-Chart Card）として実現されています。

信号シーケンス・エディタ内で『SDL 変換』コマンドを実行することで、指定の入出力主体に関する信号が SDL-GR シンボルに変換されます。

33.2 保存場所

Signal-Sequence 関連のファイルは論理パス名としての card-system:Signal-Sequence; の下で管理されています。ATR 通信システム研究所では、その絶対パス名は

- `clm01:>local>binary>signal-sequence>**>*.*.*`
- `clm13:>local>binary>signal-sequence>**>*.*.*`

のどちらかが対応します。利用されるマシンのサーバマシンにより異なります。

以下で、個々のファイルのインストール先を述べます。

1. プログラム
   card-system:signal-sequence;*.lisp
   card-system:signal-sequence;*.bin

33.3 起動方法

Signal-Sequence システムの起動

1. load system Signal-Sequence
   システム環境の設定がなされます。

2. create card system manager Any :system type Signal-Sequence
   『Signal-Sequence』用のマネージャを作成

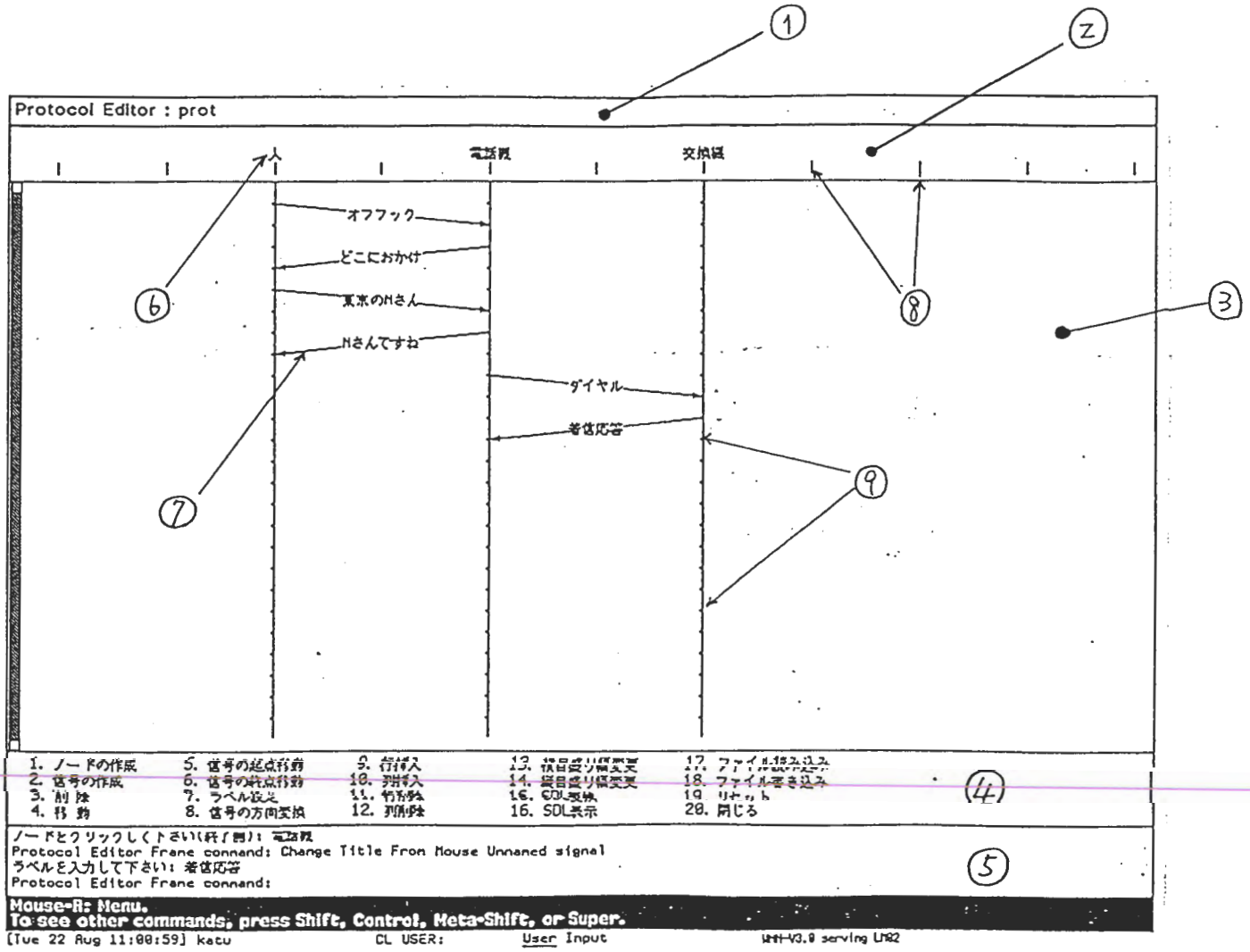
3. select + A

『Signal-Sequence』用のマネージャ画面の選択

4. Signal-Sequence-Chart Card の作成

Mouse-L でカード作成モードに入り、Signal-Sequence-Chart を選択します。その後、作成された Signal-Sequence-Chart をクリック（Mouse-L）することで Signal-Sequence-Chart エディタをオープンすることができます。

3．Ｐｒｏｔｏｃｏｌ－Ｅｄｉｔｏｒフレームの画面構成



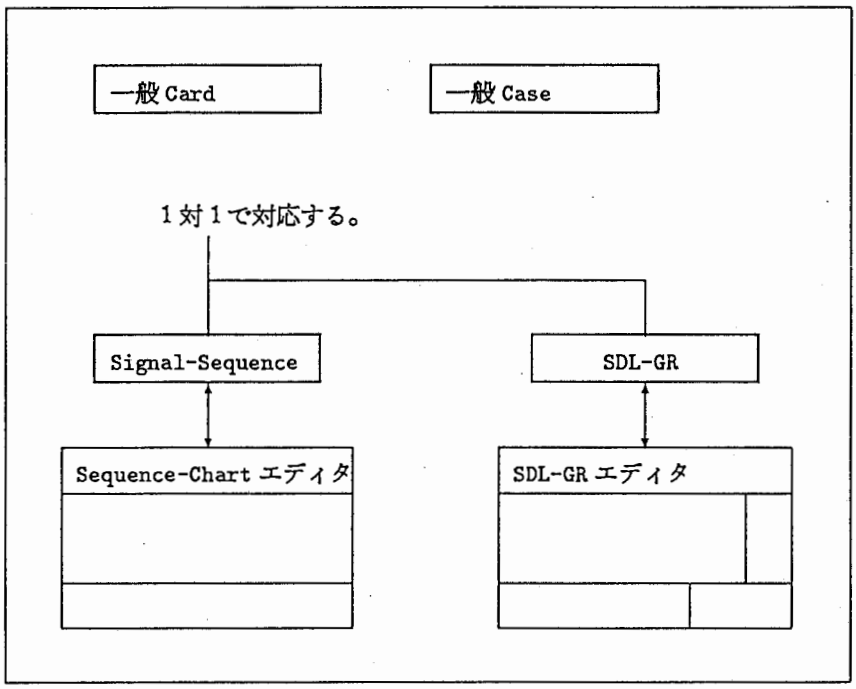
★ ウインドウ関連
- ① title ペイン
- ② node-display ペイン
- ③ display ペイン
- ④ command-menu ペイン
- ⑤ interacter ペイン

★ 表示実態
- ⑥ ノード
- ⑦ シグナル（信号）

★ 目盛り
- ⑧ 横目盛り
- ⑨ 縦目盛り

第 34 章

Signal-Sequence システムの構成



Signal-Sequence システムは上図に示す通り、カードシステム上の Card オブジェクトである Signal-Sequence-Chart と、それに 1 対 1 で対応する Sequence-Chart エディタとで構成されている。

また、Sequence-Chart エディタ内で『SDL 変換』コマンドを実行することで、SDL-GR オブジェクトとリンク付けられ、1 対 1 で対応する。

表 34.1: カードシステム上の関連オブジェクトの説明

オブジェクト名	説　　明
SDL-GR オブジェクト	本カードは SDL-GR 図を編集するために提供されている。本カードは、カードシステムで提供されている他のカードと同様、任意のケースへの格納やリンク付けの操作が可能である。
Signal-Sequence オブジェクト	本カードは信号シーケンス図を編集するために提供されている。本カードは、カードシステムで管理されている他のカードと同様、任意のケースへの格納やリンク付けの操作が可能である。

第 35 章

Signal-Sequence システムの利用方法

35.1 システムの起動

1. システムのロード
   Command: Load System Signal-Sequence の実行
2. カードシステムマネージャーの作成
   Command: Create Card System Manager any :System Type Signal-Sequence
3. カードシステムのメインウィンドウの作成（¡select¿A）

35.2 Signal-Sequence オブジェクトの作成

カードシステムのメインウィンドウにおいて

1. Create Card コマンドの選択
2. pop up 表示される Form Card メニューより Signal-Sequence-Chart を選択
3. 属性の入力（タイトル等）

35.3 Sequence-Chart エディタの Open,Close

- Open （アイコン状態 → 内容表示状態）
  Signal-Sequence オブジェクトをマウスクリック（#\mouse-1）
- Close （内容表示状態 → アイコン状態）
  Sequence-Chart エディタ内の『閉じる』コマンドを実行する。

第 36 章

Sequence-Chart エディタがもつコマンドメニュー

1. ノード作成	2. 信号作成	3. 削除	4. 移動
5. 信号の起点移動	6. 信号の終点移動	7. ラベル設定	8. 信号の方向変換
9. 行挿入	10. 列挿入	11. 行削除	12. 列削除
13. 横目盛り幅変更	14. 縦目盛り幅変更	15.SDL 変換	16.SDL 表示
17. ファイル読み込み	18. ファイル書き込み	19. リセット	20. 閉じる

1. ノード作成
   位置情報とラベル名情報からノードを作成する。

   (a) 作成位置（横目盛り）のマウス指示（#\mouse-1）。

   (b) ノードの名称（ラベル）をキーボード入力。

   ※ 作成位置に既に他のノードが存在するときは、そのノードから右にあるすべての図形を右方向にシフトした後に作成する。

2. 信号作成
   開始位置と終了位置から信号を作成する。

3. 削除
   ノード、または信号を削除する。

   (a) 削除するノード、または信号のマウス指示（#\mouse-1）。

   (b) 削除確認

4. 移動
   ノード、または信号を移動する。

   - ノードの移動

     (a) 対象ノードの選択（#\mouse-1）。

     (b) 移動位置（横目盛り）のマウス指示（#\mouse-1）。

     ※ 移動位置に既に他のノードが存在するときは、そのノードから右にあるすべての図形を右方向にシフトした後に移動する。

   - 信号の移動

     (a) 対象信号の選択（#\mouse-1）。

(b) 移動後の開始位置（縦目盛り）のマウス指示（#\mouse-1）。

(c) 移動後の終了位置（縦目盛り）のマウス指示（#\mouse-1）。

※移動位置に既に他の信号が存在するときは、その信号から下にあるすべての信号を下方向にシフトした後に移動する。

5. 信号の起点移動

信号の開始位置を変更する。

(a) 対象信号の選択（#\mouse-1）。

(b) 移動後の開始位置（縦目盛り）のマウス指示（#\mouse-1）。

6. 信号の終点移動

信号の終了位置を変更する。

(a) 対象信号の選択（#\mouse-1）。

(b) 移動後の終了位置（縦目盛り）のマウス指示（#\mouse-1）。

7. ラベル設定

ノード、または信号に名称を設定する。

(a) 対象ノード、または信号の選択（#\mouse-1）。

(b) 新名称のキーボード入力。

8. 信号の方向変換

信号の方向を逆転する。

(a) 対象信号の選択（#\mouse-1）。

9. 行挿入

位置と行数から行を挿入する。

(a) 挿入位置（縦目盛り）のマウス指示（#\mouse-1）。

(b) 行数のキーボード入力

10. 列挿入

位置と列数から列を挿入する。

(a) 挿入位置（横目盛り）のマウス指示（#\mouse-1）。

(b) 列数のキーボード入力

11. 行削除

位置と行数から行を削除する。

(a) 削除位置（縦目盛り）のマウス指示（#\mouse-1）。

(b) 行数のキーボード入力

12. 列削除

位置と列数から列を削除する。

(a) 削除位置（横目盛り）のマウス指示（#\mouse-1）。

(b) 列数のキーボード入力

13. 横目盛り幅変更
    横目盛りの幅を変更する。

    (a) 目盛り幅のキーボード入力

14. 縦目盛り幅変更
    縦目盛りの幅を変更する。

    (a) 目盛り幅のキーボード入力

15. SDL 変換
    変換処理対象（ノード）に関連する全信号から SDL 図に変換する。

    (a) ノードのマウス指示（#\mouse-1）。

16. SDL 表示
    現 Sequence-Chart エディタに対応する SDL エディタをオープンする。

17. ファイル読み込み
    ファイルに保存された信号ーシーケンス図を読み込んで画面に出力する。

    (a) 入力ファイル名のキーボード入力

    (b) 処理確認

18. ファイル書き込み
    現 Sequence-Chart エディタ上の信号ーシーケンス図をファイルに保存する。

    (a) 出力ファイル名のキーボード入力

19. リセット
    現 Sequence-Chart エディタ上の全情報をクリアする。

    (a) 処理確認

20. 閉じる
    現 Sequence-Chart エディタをクローズする。

第 37 章

現 Sequence-Chart エディタ上でのマウスジェスチャー

Sequence-Chart エディタ上では次にあげるマウス操作をサポートしている。

- 横目盛りセンス時

  left　ノード作成

- 縦目盛りセンス時

  left　信号作成

- ノードセンス時

left	信号作成
middle	ラベル設定
shift-left	移動
control-middle	削除

- 信号センス時

middle	ラベル設定
shift-left	起点移動
shift-middle	終点移動
control-middle	削除

第 38 章

UNIX パッケージ

38.1 はじめに

Symbolics は強力なプログラム開発用ツールを提供していますが、世界中で広く使用されている “UNIX システム” にもいくつか有意義なものが有ります。UNIX パッケージはそのいくつかのコマンドを Lisp の関数として実現したツール群です。

“UNIX システム” では、diff(ファイルの差分比較) や、grep(ファイルのパターン検索) は以下の様に指定します。

```
% diff file1 file2
% grep 'abc*' file
```

UNIX パッケージでは Lisp の関数として以下の様に書きます。

```
Command: (unix:diff "file1.lisp" "file2.lisp")
Command: (unix:grep :pattern "abc*" :input "file.lisp")
```

第 39 章

UNIX パッケージの起動法

39.1 機能概要

Symbolics は強力なプログラム開発用ツールを提供していますが、世界中で広く使用されている “UNIX システム” にもいくつか有意義なものが有ります。UNIX パッケージはそのいくつかのコマンドを Lisp の関数として実現したツール群です。

“UNIX システム” では、diff(ファイルの差分比較) や、grep(ファイルのパターン検索) は以下の様に指定します。

```
% diff file1 file2
% grep 'abc*' file
```

UNIX パッケージでは Lisp の関数として以下の様に書きます。

```
Command: (unix:diff "file1.lisp" "file2.lisp")
Command: (unix:grep :pattern "abc*" :input "file.lisp")
```

現在のリリース・バージョンは 2.2 です。

39.2 保存場所

unix のフィジカル・パス及びロジカル・パスは、次のとうりです。

1. フィジカル・パス
   (a) COM
   `CLM01:>local>binary>unix>`
   (b) COM2
   `CLM13:>local>binary>unix>`
2. ロジカル・パス
   `unix:unix;`

詳しくは、次のファイルを参照して下さい。

- sys;site;unix.system
- sys:site;unix.translations

39.3 起動方法

*Dynamic Lisp Listener*で次のようにします。

Command: Load System *unix*

第 40 章

基本仕様

40.1 メタキャラクタ

Symbolics ではファイル名に “*” というワイルドカードを使用することが許されていますが、“UNIX システム” には “*” 以外にも数多くの種類のメタキャラクタというものがあります。

UNIX パッケージでは以下のメタキャラクタが各関数で共通に使用できます。

40.1.1 ファイル名

*	任意の文字列 (空でもよい)
?	任意の 1 文字
[abc]	a か b か c のいづれか 1 文字
[5-9]	5 から 9 までの 1 文字
[d-y]	d から y までの 1 文字
[!abc]	a、b、c 以外の 1 文字
[{lisp}{text}]	lisp または text という文字列
\	*?[]{} の機能を無効にする

表 40.1: ファイル名のメタキャラクタ

40.1.2 パターン (grep、awk で使用)

*	直前の文字が 0 個以上
+	直前の文字が 1 個以上
.	任意の 1 文字
^	先頭
$	末尾
[abc]	a か b か c のいずれか 1 文字
[^abc]	abc 以外の 1 文字
\	*+.^$[] の機能を無効にする

表 40.2: パターンのメタキャラクタ

40.2 デフォルト・ディクトリ

Common Lisp ではパス名を “*default-pathname-defaults*” という変数にバインドされた値とマージして使用していますが、UNIX パッケージでは以下の変数を使用してパス名を作成します。

デフォルト・ディクトリの変更は unix:cd、参照は unix:pwd という関数で行います。

```
(defvar *current-pathname* (user-homedir-pathname))
```

UNIX パッケージをロードした状態ではログイン・ホーム・ディクトリがバインドされています。

40.3 関数の戻り値

UNIX パッケージの各関数は戻り値として二つの値を返します。最初の値は関数の処理結果を表すもので、正常に終了した場合は各関数の処理した値を返し、異常終了した場合は nil を返します。二つ目の値はエラーの内容を表すもので、正常に終了した場合は nil 、異常終了した場合はエラーメッセージをストリングにして返します。

40.4 関数の入力

UNIX パッケージの各関数では入力を :input というキーワード・パラメータを使用して指定します。

```
Command: (unix:wc :input "file.text")
Command: (unix:sort :input '(xab lmn abc))
```

“UNIX システム” では入力をファイル名で指定したり、キーボード入力を行わせたりしますが、UNIX パッケージでは以下の 3 つの入力方法があります。

40.4.1 ストリング

ストリングを指定された時はそれをパス名としてファイルを探しに行きます。メタキャラクタの指定が可能で、検索結果が複数になった場合は、それぞれのファイルに対して処理を行います。

(一部の関数は入力ファイルをひとつしか処理しないものがあり、その時は最初に見つかったファイルついて処理を行います。)

尚バージョン番号を指定していない場合は、特別な場合を除いて最新バージョンを指定したものと見なし処理を行います。従って最新バージョンを現す newest は指定しないで下さい。

ファイルは common-lisp の read-line で読み、ストリングとしてそれぞれの処理を行います。

40.4.2 リスト

リストが指定された時は、トップレベルの要素を処理の対象とします。尚、処理がストリングの時は以下の様に行います。

```
(abc "Abc" (xyz nil)) → ("ABC" "Abc" "(XYZ NIL)")
```

40.4.3 ストリーム

ストリームが指定された時は、ストリングの場合と同様に Common Lisp の read-line で読み、ストリングとしてそれぞれの処理を行います。

40.5 関数の出力

UNIX パッケージの各関数では出力を :output というキーワード・パラメータを使用して指定します。

Command: (unix:wc :input "file.text" :output "file-out1.text")

Command: (unix:sort :input '(xab lmn abc) :output "file-out2.text")

"UNIX システム" では出力はファイルや画面に表示させたりしますが、UNIX パッケージでは以下の 4 つの出力方法があります。

40.5.1 :output を指定しなかった時 (nil を指定した時も含む)

処理結果を戻り値としてリストで返します。(各関数によってフォーマットが異なります。)

40.5.2 ストリング

:input と同様に指定されたストリングからファイルを検索して、戻り値としてリストで返すべきリストのトップレベルの要素を 1 ラインとしてファイルに出力します。

(検索結果が複数になった場合は処理を中断します。ファイルが存在しなかった時は指定されたストリングで新しいファイルを作成します。)

40.5.3 ストリーム

ストリングと同様に戻り値としてリストで返すべきリストのトップレベルの要素を 1 ラインとして指定されたストリームを使用してファイルに出力します。

40.5.4 関数

上記の 40.5.1 から 40.5.3 以外の時は出力関数が指定されたものと見なし、40.5.2、40.5.3 の処理と同様に戻り値のリストのトップレベルの要素を関数の引数として実行し、関数の戻り値をリストにして返します。

また:output にストリングを指定し、:append というキーワード・パラメータの引き数に t を指定することによりファイルをアペンドモードで出力することができます。

第 41 章

関数の機能説明

次ページ以降に以下の関数の機能について述べます。
(機能は UNIX System V に準拠しています。)

awk	パターン操作
cat	ファイル内容のリスティング
cd	カレントディレクトリの変更
diff	ファイルの差分をとる
find	ファイルがどこにあるか調べる
grep	文字パターンを探す
head	ファイルの先頭数行を表示
ls	ファイル名の表示
make	プログラムの保守
pwd	カレントディレクトリの表示
sort	ソーティング
tail	ファイルの末尾を表示
uniq	重複行を除く
wc	語数、行数カウント

41.1 awk パターン操作

41.1.1 形式

(unix:awk &key :f :program :input :output :append)

:f フィールドを区切る文字をキャラクタで指定する。

:program 実行するプログラムを以下の方法で指定する。

1. ストリング、ストリーム
   ストリングを指定された時はそれをパス名としてファイルを、ストリームを指定された時はそれを使用して common-lisp の read を使用して処理を行います。
2. リスト
   直接実行するプログラムをリストで指定する。

:input 入力を指定する。

:output 出力を指定する。

:append アペンドモードを指定する。

41.1.2 機能

awk は指定された入力に対して、 program で指定したパターンに一致する行を探し、アクションを実行します。
program は以下の様な形式で指定します。

(パターン [アクション])

パターン

パターンには以下の 3 つの指定が可能です。

1. :begin , :end
   :begin は最初の入力行の読み取り前、:end は最後の行の読み取り後に指定されたアクションを処理します。
2. 正規表現文字列
   入力行に対してパターン検索を行わせたい時に指定します。検索結果が t の時アクションを実行します。尚、論理記号 (:or , :and , :not) を先頭に指定して以降に正規表現文字列を指定することも可能です。
3. リスト
   リストが指定されている時は、そのリストを評価して nil 以外の時アクションを実行します。

アクション

アクションはパターンを処理した結果が nil 以外の時実行されるものをリストで指定します。処理は指定されたリストを評価します。アクションを省略すると入力行が印字されます。
またプログラム内に以下の変数を使用できます。

unix:*awk-number-record* 入力行数

unix:*awk-number-field* 入力フィールド数

unix:*awk-current-line* 入力行 (ストリング)

unix:*awk-field-list* 入力行の各フィールド(ストリング)要素のリスト

unix:*awk-field-separator* フィールド区切り文字(キャラクタ)

unix:*awk-output-pointer* 出力ストリーム

41.1.3 戻り値

最後に評価したアクションが戻り値になります。

41.1.4 使用例

```
(unix:awk :program '((:begin (setq count 0))
     ("a.*" (setq count (length unix:*awk-number-line*)))
            (:end (print count)))
  :input '("abc" " def" "xyza"))
```

入力に "a" を含む要素の長さの合計を求める。結果として 7 がプリントされる。

```
(unix:awk :program '((:begin (setq count 0))
     (:not "a.*" (setq count (length unix:*awk-number-line*)))
     (:end (print count)))
  :input '("abc" " def" "xyza"))
```

入力に "a" を含まない要素の長さの合計を求める。結果として 4 がプリントされる。

```
(unix:awk :program '((:begin (setq count 0))
     ((> (length unix:*awk-current-line*) 3)
         (setq count (1+ count)))
     (:end (print count)))
  :input '("abc" " def" "xyza"))
```

入力の要素の長さが 3 より大きいもの数える。結果として 2 がプリントされる。

41.1.5 制限

1. 出力がほかの関数とは異なります。
2. ストリング及びストリームを指定したときはストリームを、unix:*awk-output-pointer* にセットするだけで関数内では出力に対する処理は行いません。リストが指定された場合は入力を全て出力関数に渡します。

41.2 cat ファイル内容のリスティング

41.2.1 形式

(unix:cat &key :s :v :t :e :input :output :append)

:s ファイルが存在しなくてもメッセージを出力しない。

:v 非印刷文字 (タブ、復帰改行、用紙送りは除く) を識別できるように出力する。

:t タブを ^[tab] で出力する。

:e 復帰改行文字を文字 $ で出力する。

:input 入力を指定する。

:output 出力を指定する。

:append アペンドモードを指定する。

41.2.2 機能

cat は指定された入力を出力します。(詳細は UNIX のマニュアルを参照して下さい。)

41.2.3 戻り値

入力にリストが指定された時はそのままリストを返す。以外は各ファイルの内容をそれぞれリストにして全体をリストにして返します。

41.2.4 使用例

(unix:cat "abc.dat")

ファイル abc.dat の内容を戻り値とする。ファイルの内容は、1行毎に文字列に変換し、リストにする。たとえば、ファイル abc.dat の内容が、

1行目
2行目
終わりの行

ならば、戻り値は、

(("1行目" "2行目" "終わりの行"))

となる。

(unix:cat '(a b c))

Common-Lisp の関数 values と同じ結果になる。

41.2.5 制限

1. :v オプション指定時の非印刷文字について
   非印刷文字は unix:*non-printing-characters-table* を使用して変換を行います。変換テーブルに存在しない文字は common-lisp の char-code を使用してコード属性を "^[コード]" の形に変換します。

2. :v , :t , :e オプションの関係
   :t オプションは:v オプションを包括し、:e オプションは:v オプション, :t オプションとは独立して指定できます。

41.3 cd カレント・ディレクトリの変更

41.3.1 形式

(unix:cd &optional ディレクトリ名)

ディレクトリ名 新しいディレクトリ名を指定する。

41.3.2 機能

cd はカレント・ディレクトリ (unix:*current-pathname*) の変更を行います。ディレクトリ名が指定されていないときは、ログイン・ホーム・ディクトリがセットされます。

指定されたディレクトリと unix:*current-pathname* をマージして新しいディレクトリ名を作成します。またディレクトリ名に正規表現文字列も指定可能です。

41.3.3 戻り値

新しいパス名をかえします。

41.3.4 使用例

ホーム・ディレクトリが "CLM01:>unix>" の時、次のようになります。

- (unix:cd "dir1")
  "CLM01:>unix>dir1>"
- (unix:cd " dir2")
  "CLM01:>unix>dir2>"
- (unix:cd)
  "CLM01:>unix>"
- (unix:cd "*1")
  "CLM01:>unix>dir1>"

41.3.5 制限

1. カレント・ディレクトリをひとつ上がる場合の指定
   先頭に ">" を上がる個数付けて指定する。
2. ホーム・ディレクトリの下のサブ・ディレクトリを指定する場合
   "~> サブ・ディレクトリ " の形で指定します。
3. 正規表現文字列
   指定された文字列からディレクトリを検索してセットします。(複数見つかった時は処理を中断します。)

41.4 diff ファイルの差分をとる

41.4.1 形式

(unix:diff 入力 1 入力 2 &key :b :output :append)

<入力 1,2> 入力を指定する。(:input と同じものが指定できます。)

:b 行または文字列の後ろのブランクを無視する。

:output 出力を指定する。

:append アペンドモードを指定する。

41.4.2 機能

diff は指定された入力の内容を一致させるために変更する必要のある行を出力します。(詳細は UNIX のマニュアルを参照して下さい。)

41.4.3 戻り値

変更内容を示すいくつかのリストを、ひとつのリストにして返します。変更内容を示すリストは以下の形式です。

(変更内容　入力 1 の該当リスト　入力 2 の該当リスト)

変更内容は以下の 3 つの形式があります。

n1 a n3 n4
n1 n2 d n4
n1 n2 c n3 n4

41.4.4 使用例

(unix:diff '(a b c d e) '(a c x e))

戻り値は、(("2 d 2" (b) nil) ("4 c 3" (d) (x))) となります。

41.4.5 制限

特に無し。

41.5 find ファイルがどこにあるか調べる

41.5.1 形式

(unix:find パス名 式)

パス名 検出の起点となるディレクトリ名を指定する。

式 以下の指定が行える。

:name ファイル名
　ファイル名 (正規表現) とマッチするならば真。

:type-dir ファイルがディレクトリならば真。

:user ユーザ名
　ファイルの持ち主が同じならば真。

:size サイズ [:c]
　ファイルのサイズが指定されたブロック数よりも大きければ真。c が指定されているならば、文字数で計る。

:time 日時
　ファイルが指定された日時の間に更新されたものならば真。
　time after "1987.10.21 17:00" time befor "1987.10.21"

:exec 指定したコマンドまたは関数を実行し、その戻り値が真の場合に真となる。対象ファイル名は、で示す。

:ok コマンドを実行する前に確認メッセージを出し、y と応答した場合にのみコマンドを実行する。

:print 常に真。ファイル名をプリントする。

:newer ファイル名
　ファイルが指定されたファイルよりも新しければ真。

:newest バージョン内でいちばん新しければ真。

:older ファイル名
　ファイルが指定されたファイルよりも古ければ真。

:oldest バージョン内でいちばん古ければ真。

:or 式の論理和をとる。

:not 式の否定をとる。

:and 式の論理積をとる (省略時の値)。

41.5.2 機能

find は指定されたパス名についてディレクトリ階層を再起的に下降し、式に書かれたものと一致するファイルを探します。(詳細は UNIX のマニュアルを参照して下さい。)

41.5.3 戻り値

ファイル名をリストにして返します。

41.5.4 使用例

```
(unix:find "clm08:>takashi>tools"
           '(:name "*.lisp" :time :after "1987-10-1" :print))
```

'("formatting-card.lisp.8" "formatting-card.lisp.9" "graph-3.lisp.15")

1987-10-1 以降に作成したリスプ・ファイルを取り出す。

```
(unix:find ">takashi"
         '((:or :name "*.lisp" :name "*.bin")
     :time :after "1987.10.1" :print))
```

ファイル名が *.lisp か *.bin で 1987.10.1 以降に作成したものを取り出す。

```
(unix:find ">takashi" '((:name "*.lisp" (:not :newest) :ok "Delete File {}")))
```

最新バージョン以外の *.lisp ファイルを削除する。

41.5.5 制限

1. :name について
   ディレクトリ付きのファイル名は指定できません。
2. :time
   日時はストリングで指定して下さい。指定されたストリングを time:parse-universal-time という関数で処理します。

41.6 grep 文字パターンを探す

41.6.1 形式

(unix:grep &key :v :c :l :n :i :s :w :f :choose :pattern :input :output :append)

:v パターンを含む行を除いた行または要素を出力する。

:c 一致したパターンを含む行または要素の数を出力する。

:l 一致したパターンを含むファイル名を出力する。

:n 各出力の先頭にファイル、リスト内の行または要素番号を入れる。

:i 大文字と小文字とを区別しない。

:s エラー・メッセージを出力しない。

:w パターンと単語とをマッチングする。

:f パターンの入っているファイル名を指定する。

:choose マッチング対象を取り出す関数を指定する。(default:values)

:pattern マッチング・パターンを正規表現文字列で指定する。

:input 入力を指定する。

:output 出力を指定する。

:append アペンドモードを指定する。

41.6.2 機能

grep は指定された入力をパターン検索して、そのパターンを含むものを出力します。(詳細は UNIX のマニュアルを参照して下さい。)

41.6.3 戻り値

出力に対するオプションの :c , :l , :n が指定されていないときは、パターンを含む行をリストにして返します。(入力がひとつの時)

:c , :l , :n が指定されて入るときは、以下の様な形式をリストにして返します。

:n が指定された時 ([ファイル名] 行番号 行の内容)
:c が指定された時 ([ファイル名] マッチした行数)

41.6.4 使用例

```
(unix:grep :pattern "abc.*xyz" :input "*.[{text}{lisp}]")
```

ファイル (*.text,*.lisp) から 1 行単位にテキストを入力し、マッチング・パターンに一致する行をリストにし、関数の戻り値とする。

```
(unix:grep :pattern "abc" :input '("ABCabc" (abcd nil) (nil "abxc")))
```

("ABCabc" (abcd nil))

```
(unix:grep :n :pattern "xyz" :input '("xyz" (abcd xyz) xyz))
```

((0 "xyz") (1 (abcd xyz)) (2 xyz))

41.6.5 制限

1. :c , :l , :n の関係

   :c , :l は、:n と同時に指定できません。(:n を無視して処理を続行します。)

2. :l :l は入力がリストの時は指定できません。(:l を無視して処理を続行します。)

41.7 head ファイルの先頭数行を表示

41.7.1 形式

(unix:head &key :- :input :output :append)

:- 表示する行または要素の数を指定する。(デフォルトは 10)

:input 入力を指定する。

:output 出力を指定する。

:append アペンドモードを指定する。

41.7.2 機能

head は指定した入力の最初の部分を出力します。(詳細は UNIX のマニュアルを参照して下さい。)

41.7.3 戻り値

入力の先頭をリストにして返します。入力が複数ファイルの時、ファイル名を示すストリングを区切りに入れます。

41.7.4 使用例

```
(unix:head :- 3 :input "abc.dat")
```

戻り値は、("一行目" "二行目" "三行目") となります。

41.7.5 制限

特に無し。

41.8 ls ファイル名の表示

41.8.1 形式

(unix:ls &key :r :a :d :c :x :m :l :g :reverse :t :f :name :output :append)

:r すべてのサブディレクトリの内容を再帰的に出力する。

:a すべてのエントリ (古いバージョン、削除マーク付きファイルも含む) を出力する。省略時は、最新バージョンだけを出力する。:l と組み合わせることにより、ディレクトリの状態を見れる。

:c マルチカラム形式で、エントリ名は縦方向にソートされる。

:x マルチカラム形式で、エントリ名は横方向にソートされる。

:l ファイルの状態をロング形式で出力する。

1. 所有者名
2. ファイル・サイズ
3. 作成日時
4. バックアップ状態
5. ファイル名
6. バージョン番号
7. 削除マーク

:g 所有者名を出力しない。

:reverse 逆順にソートする。

:t 時間順にソートする。

:f ファイルがディレクトリならば、¿を名前の後ろにつける。

:name 表示したいファイル名またはファイル名のリスト (正規表現が可能)。

:output 出力ファイルを指定する。

:append アペンドモードの指定。

41.8.2 機能

ls は指定されたディレクトリの内容を出力します。(詳細は UNIX のマニュアルを参照して下さい。)

41.8.3 戻り値

出力に対するオプションの :r , :l , :g が指定されていないときは、ファイル名をリストにして返します。
:r , :l , :g が指定されて入るときは、以下の様な形式をリストにして返します。

:l , :g が指定された時 ([所有者名] ファイル・サイズ . . .)
:r が指定された時 (ディレクトリ名 (出力形式リスト))

41.8.4 使用例

```
(unix:ls :a t :l t)

(("unix" 123 "1988-02-28 14:00:02" nil "abc.dat.2" 2 "del")
 ("unix" 434 "1988-01-28 17:30:33" nil "dsw.text.3" 3 nil))
```

41.8.5 制限

1. :c , :x オプションについて

   :c , :x は出力にファイルかまたはストリームを指定した時のみ有効です。

41.9 make プログラムの保守

41.9.1 形式

(unix:make &key :target :script :print :execute :touch)

:terget ターゲット名をストリングで指定する。

:script 処理を行うスクリプトを指定する。(省略したときは"make-script-lisp"を使用する。)

:print 実行するコマンドを表示するかどうかを t nil で指定する。
(デフォルトは t)

:execute コマンドを実行しターゲットの更新を行うかどうかを指定する。
(デフォルトは t)

:touch コマンドを実行せずにターゲットの日時だけを変更するかどうかを指定する。
(デフォルトは nil)

41.9.2 機能

make は指定されたターゲット名とスクリプトから、ターゲットと関連ターゲットの各要素の更新日時を比較してターゲットが古いときスクリプトに指定されたコマンドを実行します。

スクリプトは以下の様な形式で書き、ターゲット名、関連ターゲット名はストリングで指定します。またリストで指定することも可能です。コマンド部分はストリングか関数をリストで指定します。

(ターゲット名　関連ターゲット名　コマンド)

SUFFIXES によるターゲットの推測機能があります。SUFFIXES は以下の様に宣言しているため、ユーザの独自の依存関係も追加できます。

```
(defvar *unix-make-suffixes*
    '(("bin" "lisp" ":lisp-compile :terget-name.lisp :compile-key))
```

スクリプト内でマクロを以下の形式で定義することができます。スクリプト中では: マクロ名 の形で指定します。

(:set マクロ名　展開形)

あらかじめ以下のマクロが定義されています。

:target-file 現在のターゲット名

:terget-name 現在のターゲット名から SUFFIXES を除いたもの

:newer-relation 現在のターゲットより新しい全ての関連ターゲット

:command-object コマンドの実行の対象となった関連ターゲット

:lisp-compile リスプのコンパイルコマンドの "compile file"

:compile-key

41.9.3 使用例

```
(unix:make :terget "all"
   :script '(("all" ("foo.bin" "bar.bin" "baz.bin")
                    (format t "~%Done."))
     (("foo.bin" "bar.bin") "macros1.lisp"))
     ("baz.bin" "macros2.lisp"))
```

実行結果は以下の通りです。

```
compile file foo.lisp
compile file bar.lisp
(format t "~%Done.")
Done.
```

41.9.4 制限

1. :execute , :touch は同時に指定できません。
2. :script は :input と同様な指定が行えます。
3. 関連ターゲットに指定されたファイルが存在しないときエラーとなります。

41.10 pwd カレント・ディレクトリの表示

41.10.1 形式

(unix:pwd)

41.10.2 機能

pwd はカレント・ディレクトリのパス名を表示します。

41.10.3 戻り値

カレント・ディレクトリのパス名を返します。

41.10.4 使用例

(unix:pwd)
戻り値は、"CLM01:>unix>" となります。

41.10.5 制限

特に無し。

41.11 sort ソーティング

41.11.1 形式

(unix:sort &key :c :m :u :d :f :month :n :r :b :t :i :input :output :append)

:c 入力がソート済みかチェックする。

:m ソート済みの入力をマージする。

:u 同一行・要素をユニークにする。

:d 辞書順にソートする。

:f 大文字、小文字の区別をしない。

:i 非数値比較において ASCII コードの (040 ～ 0176) 以外の文字を無視する。

:month 月名として比較する。("Jan" < "Feb" < "Dec")

:n 数値として比較する。

:r 逆順にソートする。

:b フィールド内の先行ブランクを無視する。

:t '(c list) c の文字をフィールド区切り記号とし、ソートキーを list で指定する。:t '(#\: ((0 0 2 0)(5 0))) 区切り記号は、∈ B苔キーは、0 フィールドの 0 文字目から、2 フィールドの 0 文字の直前まで。第 2 キーは、5 フィールドの先頭から最後まで。

:i 非数値比較において ASCII コードの (040 ～ 0176) 以外の文字を無視する。

:input 入力を指定する。

:output 出力を指定する。

:append アペンドモードを指定する。

41.11.2 機能

sort は指定された入力をソートします。(詳細は UNIX のマニュアルを参照して下さい。)

41.11.3 戻り値

処理した入力行をリストにして返します。(入力ファイルが複数でもひとつにまとめて処理を行います。)

41.11.4 使用例

```
(unix:sort :input '(def bde aed xye) :r t)
```

戻り値は、(xye def bde aed) となります。

41.11.5 制限

1. :n , :month 関係
   :n , :month は同時に指定できません。(:month を無視して処理を続行します。)

2. :input にリストが指定された場合
   リストのトップレベルの要素をストリングにして処理を行います。

41.12 tail ファイルの末尾を表示

41.12.1 形式

(unix:tail &key :+ :- :c :input :output :append)

:+ 出力の開始位置を入力の先頭からの距離で指定する。

:- 出力の開始位置を入力の末尾からの距離で指定する。(デフォルトは 10)

:c n を文字数を単位として計算する。

:input 入力を指定する。

:output 出力を指定する。

:append アペンドモードを指定する。

41.12.2 機能

tail は指定された入力の最後の部分を出力します。(詳細は UNIX のマニュアルを参照して下さい。)

41.12.3 戻り値

処理した入力行をリストにして返します。

41.12.4 使用例

入力ファイル "abc.dat" が n 行の時の処理。

```
(unix:tail :- 3 :input "abc.dat")
```

戻り値は、("n-2 行目" "n-1 行目" "n 行目") となります。

41.12.5 制限

1. 入力が複数の時最初に見つかったファイルのみ処理を行います。

41.13 uniq 重複行を除く

41.13.1 形式

(unix:uniq &key :u :d :c :- :+ :input :output :append)

:u 重複していない行だけを出力する。

:d 重複した行の1行だけを出力する (省略時は :u :d)。

:c 各行の前に重複の回数を表示する (:u と:d に優先する)。

:- 各行の最初のnフィールドが、前のブランクとともに無視される。

:+ 各行の最初のn文字を無視する。:n 指定があれば、nフィールド分スキップした後にn文字スキップする。

:input 入力を指定する。

:output 出力を指定する。

:append アペンドモードを指定する。

41.13.2 機能

uniq は指定された入力の、隣接する行を比較して、重複する行を削除します。(詳細は UNIX のマニュアルを参照して下さい。)

41.13.3 戻り値

処理した入力行をリストにして返します。

41.13.4 使用例

入力ファイル "abc.dat" が "abc","abc","dge","abc" の時の処理。

```
(unix:uniq :input "abc.dat")
```

戻り値は、("abc" "dge" "abc") となります。

41.13.5 制限

1. :input にリストが指定された場合、リストの第1レベルの要素を string 化したものを処理対象とする。
2. :input に複数ファイルが指定された場合、最初の1ファイルについてのみ処理する。

41.14 wc 語数、行数カウント

41.14.1 形式

(unix:wc &key :l :w :c :input :output :append)

:l 行数または第 1 レベルの要素数を出力する。

:w ワード数を出力する。

:c 文字数を出力する (省略時は :l :w :c)。

:input 入力を指定する。

:output 出力を指定する。

:append アペンドモードをしていする。

41.14.2 機能

wc は指定された入力の行、ワード、文字を数えます。(詳細は UNIX のマニュアルを参照して下さい。)

41.14.3 戻り値

:l , :w , :c がいずれも指定されていないの場合は、以下の様な形式をリストにして返します。

(行数 語数 文字数 {ファイル名})

:l , :w , :c がいずれかひとつに t だけが指定されていないの場合は、以下の様な形式をリストにして返します。

[行数 | 語数 | 文字数]

41.14.4 使用例

```
(uxnix:wc :input '("UNIX operating system is developed and licensed by ATT"
                   "There are two primary characteristics of a flavor:"))
```

(2 17 104)

```
(unix:wc :l nil :w t :c nil :input "sample.data" :output "sample.wc")
```

(10)

ファイル sanple.data の内 one two three four five six seven eight nine ten ならは、ファイル sample.wc には、10 が出力される。

41.14.5 制限

1. :input にリストが指定された場合の処理について
   行数はトップレベルの要素数、ワード数はリスト中の空白、タブをワードの区切りと見なして計算し、語数は要素中の " ( ) も 1 文字とし、nil は 3 文字として計算します。

第 42 章

UNIX パッケージのインストレーション

UNIX パッケージはシステムとして定義されていますから、次のコマンドを発行することにより、利用することが可能となります。

Command: Load System UNIX

(詳細は、 “sys:site;unix.system”、 “sys:site;unix.translations” を参照して下さい。)

付録 A

UNIX パッケージのライブラリ

次の関数が UNIX パッケージから Export されています。

A.1 regexp-step パターン・マッチングを行う

A.1.1 形式

(unix:regexp-step pattern-list source-list &optional ignore-code)

pattern-list パターンのリストを指定する。

source-list マッチングの対象となる文字列をリストで指定する。

ignore-code 大文字・小文字の区別をしない。

機能

pattern-list(正規表現文字列を内部表現形式に展開したリスト) と、source-list のパターン・マッチングを行います。

A.2 regexp-file-compile 正規表現文字列で指定されたファイル名の展開

A.2.1 形式

(unix:regexp-file-compile pattern &optional (silent-mode t))

pattern ファイル名を指定する。

silent-mode エラー・メッセージを出力しない。

A.2.2 機能

正規表現文字列で指定されたファイル名を内部表現形式に展開する。

A.3 regexp-compile 正規表現文字列で指定された文字列の展開

A.3.1 形式

(unix:regexp-compile pattern &optional (silent-mode t))

pattern 文字列を指定する。

silent-mode エラー・メッセージを出力しない。

A.3.2 機能

正規表現文字列で指定された文字列を内部表現形式に展開する。

付録 B

Chart Editor パッケージ

B.1 機能概要

簡単な図形編集エディタの “Chart Editor” を開発しました。以下にこのシステムの概要と簡単な操作説明を述べます。

図形編集エディタ、 “Chart Editor” の今回のバージョンでサポートする機能概要を以下に述べます。

1. サポートする図形オブジェクト
   - ボックス
   - ボックスとボックスを結ぶアーク
   - ライン
2. 図形の配置
   - グリッドのサポート
3. “Card System” とのリンク
   - 作成したボックスと “Card System” のオブジェクトと関係付けられる。

B.2 Chart Editor の図形オブジェクト

今回のバージョンでサポートする図形オブジェクトは以下のものです。

1. ボックス
2. ボックスとボックスを結ぶアーク
3. ライン

B.2.1 ボックス

ボックスの属性は次のものをサポートします。

- ボックスの形状
  “Card System” のアイコン・形状と同じものが作成できます。
  また、形状に Nil を指定することで、ラベルのみの表示を実現できます。

- ボックスの大きさ
  縦、横の大きさをピクセルで指定します。

- 線種及び線の幅
  実線、破線のいずれかを指定します。

- 中塗りのパターン
  システムのデフォルト値を使用していますので、この中から選択します。

- ボックス・ラベル
  ボックスの中央に文字列(ラベル)を表示させることができます。またキャラクタ・スタイルを指定することで、ユーザが作成したフォントで文字列を表示することができます。

- アークの入出力位置
  今回のバージョンではサポートしません。

ボックスの形状と大きさの関係を図 B.1 に示します。

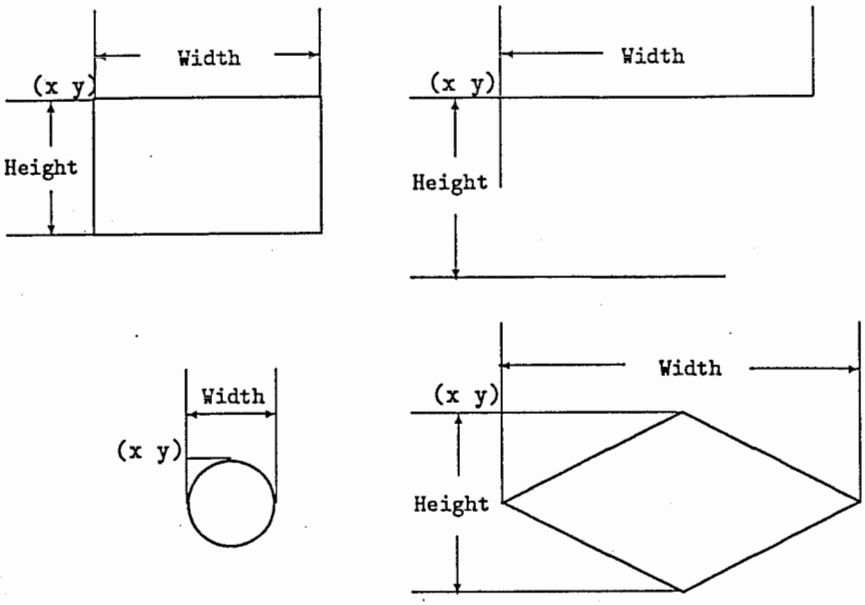


図 B.1: ボックスの形状と大きさの関係

B.2.2 アーク

アークとはボックスとボックスを結ぶ線のことです。そのアークの属性を以下に示します。

1. 方向
   アークを作成する時に二つのボックスを指示した順で方向が決まります。作成後に変更することもできます。

2. 線種
   実線、破線のいずれかを選択できます。

3. 線の幅

4. アークのボックスへの入出力位置
   今回のバージョンではサポートしません。

5. アークのラベル
   今回のバージョンではサポートしません。

今回のバージョンではアークは直線とし、編集はできないものとします。

B.2.3 ライン

任意の地点に線を描くことができます。そのラインの属性を以下に示します。

- 方向
  ラインを作成する時に指示した2点の順で方向が決まります。作成後に変更することもできます。

- 線種
  実線、破線の二種類から選択します。また矢印も設定できます。

- 線の幅
  ピクセルで指定します。

- ラインのラベル
  今回のバージョンではサポートしません。

B.3 Chart Editor 上でのコマンド

B.3.1 環境設定

"Chart Editor" の環境設定を行なうための属性は、今回のバージョンでは以下のものがあります。

1. 編集モードの切替え
   このモードがオンの時のみオブジェクトの作成、変更等が行なえます。

2. グリッド表示の切替え
   グリッドの表示をコントロールすることができます。

3. オブジェクト作成時のデフォルト値の変更
   オブジェクトはこのデフォルト値を使用して作成されます。従ってユーザは必要に応じて属性変更コマンドを使用して属性変更を行ないます。

B.3.2 オブジェクトの作成

環境変数の編集モードがオンの時、マウスでオブジェクトの作成が簡単に行なえるようにしてあります。

- ボックスの作成
  何も表示されていない位置で、Mouse-L で作成できます。

- アークの作成
  ボックスを掴んで、Mouse-M で作成できます。

- ラインの作成
  何も表示されていない位置で、Mouse-M で作成できます。

B.3.3 属性変更

オブジェクトの属性を変更するコマンドは、Meta-Mouse-L で起動させることができます。

B.3.4 移動

ボックスの移動を行なうコマンドは、Shift-Mouse-L で起動させることができます。今回のバージョンではボックスの単体移動のみを、サポートします。

B.3.5 削除

オブジェクトの削除コマンドは、Control-Mouse-L で起動されます。

B.3.6 今回のバージョンでは開発しないコマンド

今回のバージョンは暫定版であるため、次のコマンドは開発していません。

- Undo 及び Redo
- トレース
- カレント・オブジェクトを決めてコマンドを実行する方式
- 図形のグループ化
- 図形の集団移動
- 図形のコピー
- ボックス・サイズのマウスによる変更コマンド
- アークの編集
- データのファイルへのロードとファイルからのアップロード

B.4 Card System とのリンク

"Chart Editor" で作成したボックス・オブジェクトを任意の時点で、"Card System" のオブジェクトと関係付けられるようなっています。

これは、ボックスの属性変更で指定します。このとき作成できるオブジェクトは、"Card System" に登録されているカード、ケースから選択できます。

B.5 システムの利用方法

B.5.1 システムのロード

"Chart Editor" はシステムとして定義されていますので、以下のコマンドでロードして下さい。

Command: Load System chart-editor

"Card System" も同時にロードされます。

既に "Card System" がロードされている時は、次のコマンドを実行してから、上記のコマンドを入力して下さい。

Command: Load File sys:site;card-system.translations

B.5.2 Card System への登録

"Chart Editor" を利用するためには、カード・システムのマネージャに、オブジェクトを登録する必要があります。これは以下の関数を使用します。

(chart-editor::add-chart-editor-system-objects-to-card-system *manager*)

"Chart Editor" は、カード・システムのケースとして実現されています。

付録 C

Chart Editor の起動法

C.1 機能概要

図形編集エディタ、"Chart Editor" でサポートする機能概要を以下に述べます。

1. サポートする図形オブジェクト
   - ボックス
   - ボックスとボックスを結ぶアーク
   - ライン
2. 図形の配置
   - グリッドのサポート
3. "Card System" とのリンク
   - 作成したボックスと "Card System" のオブジェクトと関係付けられる。

C.2 保存場所

chart-editor のフィジカル・パス及びロジカル・パスは、次のとうりです。

1. フィジカル・パス
   (a) COM
   `CLM01:>local>binary>chart-editor>beta>`
   (b) COM2
   `CLM13:>local>binary>chart-editor>beta>`
2. ロジカル・パス
   `card-system:chart-editor;`

詳しくは、次のファイルを参照して下さい。

- sys;site;chart-editor.system
- sys:site;card-system.translations

C.3 起動方法

*Dynamic Lisp Listener*で次のようにします。

Command: Load System *chart-editor*

次にカードシステム・マネージャを作成して下さい。

Command: Create Card System Manager *manager-name* :User Name *user* :Pathname *pathname* :System Type *system-type*

chart-editor を利用するためには、カード・システムのマネージャに、オブジェクトを登録する必要があります。これは以下の関数を使用します。

(chart-editor::add-chart-editor-system-objects-to-card-system *manager*)

chart-editor は、カード・システムのケースとして実現されています。