HOKKAIDO UNIVERSITY

Title	設計生産コミュニケーションモデルの記述表現及び応用に関する研究
Author(s)	菊地, 慶仁
Degree Grantor	北海道大学
Degree Name	博士(工学)
Dissertation Number	乙第5079号
Issue Date	1996-12-25
DOI	https://doi.org/10.11501/3119591
Doc URL	https://hdl.handle.net/2115/51394
Type	doctoral thesis
File Information	000000304781.pdf

Hokkaido University Collection of Scholarly and Academic Papers : HUSCAP

設計生産コミュニケーションモデルの記述表現及び応用に関する研究

菊地慶仁

設計生産コミュニケーションモデルの記述表現及び応用に関する研究

菊地慶仁

目 次

1 緒 論 1

1.1 本論文の背景 1
1.2 設計生産活動における問題点 2
1.2.1 設計生産活動の同時並行化における問題点 2
1.2.2 設計生産における既存の研究の限界 4
1.3 本論文の目的・範囲・構成 6
1.3.1 目 的 6
1.3.2 範囲 7
1.3.3 構 成 8
1.4 本論文の特徴 9
1.5 謝 辞 14
1.6 本論文で用いる主な記号 15

2 設計生産活動における問題点の詳細化と解決の方法論 17

2.1 設計生産活動と活動に関与する対象とを表現する基本モデル 17
2.2 設計生産活動を表現する形式仕様記述言語 19
2.3 広範な活動を包含する設計生産活動統合モデル 21
2.4 形式仕様記述言語による設計生産活動表現を活用する計算機支援システムの提案 . . 22
2.5 結 論 23

3 設計生産活動の表現に関する基本モデル 25

3.1 設計生産活動を表現する基本モデルに関する分析 25
3.1.1 設計生産活動を形式仕様記述言語を用いて表現する既存の研究 26
3.1.2 本論文における設計生産活動に関する基本モデルへの課題 30
3.1.3 オブジェクト指向，プロセス代数及び通信プロトコル記述言語の特徴 32
3.2 本論文で提案する基本モデル 40
3.2.1 基本モデルの概要 40
3.2.2 プロセス代数／オブジェクト指向に基づく定式化 46
3.3 並列処理言語としての機能比較 54
3.4 特徴及び他のモデルとの比較 54
3.4.1 本論文における基本モデルの特徴 54
3.4.2 他のモデルとの比較 56
3.4.3 本論文での表現の限界 57
3.5 結 論 58

4 設計生産プロセス記述言語 60

4.1 設計生産活動を表現する言語技術に関する分析 60
4.1.1 設計生産活動を表現する形式仕様記述言語に関する問題点 60
4.1.2 本論文における設計生産活動に関する言語技術への課題 67
4.1.3 CAD データ／製品モデルデータ交換の現状 68
4.1.4 設計生産以外の領域における形式仕様記述言語の活用 70

4.2 本論文で提案する設計生産プロセス記述言語 77
4.2.1 言語の概要及び基本構造 77
4.2.2 プロセス及び入出力対象の入出力仕様表現 81
4.2.3 その他の言語仕様 85
4.2.4 モデル化の例 86
4.3 特徴及び他の言語との比較 89
4.3.1 本論文における設計生産プロセス記述言語の特徴 89
4.3.2 他の言語との比較 90
4.3.3 本論文で提案された方法での表現の限界 92
4.4 結 論 93

5 設計生産活動統合モデルの構築 95
5.1 設計生産活動の全体を表現するモデル構造に関する分析 95
5.1.1 対象モデルの範囲 95
5.1.2 対象モデルの特徴 97
5.1.3 設計生産の活動を全体を表現するモデル構造に関する問題点 98
5.1.4 本論文におけるモデル構造への課題 103
5.2 本論文で用いるモデルの全体構造 104
5.2.1 モデル間の継承構造：部品化によるリソース構造 105
5.2.2 プロセスを起動する際の動的な構造：複数プロセスの並行動作の表現 106
5.2.3 入出力の基本構造：物資の搬送及び加工作業などの表現 107
5.2.4 入出力の基本構造：入出力対象のグループ化及び階層構造の表現 108
5.3 設計生産活動統合モデルの構築 108
5.3.1 設計生産活動統合モデルの全体構造 108
5.3.2 設計グループ活動モデル 109
5.3.3 生産活動モデル 117
5.4 特徴及び他のモデルとの比較 121
5.4.1 本論文におけるモデル構造の特徴 121
5.4.2 他のモデルとの比較 122
5.4.3 複数プロセスの協調についての考察 123
5.5 結 論 123

6 プロセス管理機能実装の検証 126
6.1 設計生産プロセスの実装のための用件と分析 126
6.1.1 設計生産活動の形式表現を扱うための機能分析 126
6.1.2 計算機支援システムに必要な機能のまとめ 130
6.1.3 本論文における課題 132
6.2 計算機支援システムの提案 133
6.2.1 計算機支援システムの基本構造 133
6.2.2 プロセス実行管理機能の定式化 134
6.3 プロセス管理機能実装の検証 135
6.3.1 シミュレーションシステムの目的 135

6.3.2 シミュレーションシステムのクラス階層構造 136
6.3.3 プロセス実行管理機能の実現 137
6.3.4 プロセス管理機能の検証実験 140
6.4 特徴及び他のモデルとの比較 143
6.4.1 本論文におけるモデル構造の特徴 143
6.4.2 実際の実装に関しての考察 143
6.4.3 他の研究との比較 145
6.5 結 論 146

7 結 論 **149**

A 付 録：EPDL 言語仕様 **156**

B 付 録：設計グループ活動モデルの EPDL による記述 **178**

C 付 録：生産活動モデルの EPDL による記述 **191**

D 付 録：設計グループ活動モデルと生産活動モデルとの協調活動の EPDL による記述 **196**

図一覧

1 本論文が対象とする設計生産活動の概要 3
2 PDM の主な機能と構造 4
3 SADT 手法における基本構造 6
4 本論文の構成 9
5 提案された方法論の摘要によって期待される効果 13
6 並行活動の概念 18
7 CCS でのプロセスの入出力ポート 33
8 CCS での２つのプロセスの相補アクション 34
9 CCS での相補アクションの制限 35
10 情報入出力の過程をプロセスして扱う基本モデル 41
11 設計活動を例とした入出力対象とプロセスの表現 42
12 設計活動を例とした並行活動プロセスの表現 42
13 プロセス内部でのイベント処理の概念 43
14 基本モデルに基づく活動の記録 44
15 プロセスマネージャの概念 47
16 FDL の構文定義 (一部) 62
17 MSS 言語の構文定義 (ドメイン定義部分) 64
18 EXPRESS-P における言語拡張部分，インターフェース定義，チャンネル定義 . . . 66
19 STEP 規格の構造 70
20 LOTOS におけるプロセスの一般形 73
21 EPDL によって表現される対象と表現可能な領域 77
22 EPDL と基本モデルの対応 79
23 表現される活動モデルの構造 81
24 EPDL による表現例 86
25 EXPRESS における継承モデル 99
26 EXPRESS 継承機能を用いたモデリングの例 100
27 入出力対象をリソースとして分類したモデル化 106
28 単一のプロセス定義による複数活動の表現 107
29 設計グループの活動内容と設計グループ活動モデルとの対応 109
30 設計グループ活動モデルの構造 110
31 階層化された設計グループにおける権利の構造 112
32 仮想的設計グループの階層の実現 112
33 設計グループ活動モデルの動作例 115
34 設計グループ活動モデルの記述例 116
35 生産活動モデルの概要 118
36 生産活動モデルの構造 118
37 階層構造化された設備グループの動作例 121
38 CIMOSA における IDL から IDS への流れ 127
39 本論文で提案する計算機支援システムの一般形 133
40 プロセスマネージャの機能 134

41　シミュレーションシステムのクラス階層構造 136
42　プロセス生成部の Smalltalk プログラムの例 137
43　プロセスの生成 138
44　プロセスと ProcessManager の通信メカニズム 139
45　シミュレーションシステムの実行状況 141
46　シミュレーションシステムの画面出力例 142

表一覧

1　本論文で用いる主な記号 (1) 15
2　本論文で用いる主な記号 (2) 15
3　本論文で用いる主な記号 (3) 15
4　本論文で用いる主な記号 (4) 16
5　本論文で用いる主な記号 (5) 16
6　活動表現の観点から見た既存の研究及び本論文の評価 57
7　COSL におけるキーワードとフィールド記述 65
8　言語仕様の観点から見た既存の研究及び本論文の評価 92
9　動的設計プロセスモデルに発生するイベントとプロセス内部の反応 113

1 緒 論

本章は，設計生産コミュニケーションモデルの記述表現及び応用に関する報告を行うに当たり，研究の全体的な背景と問題点，および本論文の目的と概要とについて述べる．本章の構成は，初めに 1.1において本論文の背景について述べる． 1.2以下では，設計生産の活動に関する問題点の分析を行い， 1.3で本論文の目的・構成について， 1.4では本論文の特徴について述べる．1.5では本論文をまとめるに当たって御助力を頂いた方々に対する謝辞を述べる．さらに本論文で用いる主な数学記号を 1.6に列挙する．

1.1 本論文の背景

本節では，設計生産の活動を効率化を目的として設計生産の活動に関して起こった環境の変化について分散された環境での設計生産と中間的なデータモデルによる製品モデルデータ交換２点に関して述べる．

分散された環境での設計生産

近年の設計生産の形態として，

- 品質の向上
- ライフサイクルコストの低減
- 開発リードタイムの短縮化

を目的として，Cleetus [Cle92] によるコンカレントエンジニアリング (Concurrent Engineering) 又 Winner [WPBS88] によるサイマルテニアスエンジニアリング (Simuletaneous Engineering) が提唱されている．これらの研究は，設計や生産の各作業を短縮して効率化するだけでなく，可能な限り並行して進めることで相互の関連部分も含めて効率化を達成する概念を提案している．

コンカレントエンジニアリングは Cleetus [Cle92] によって提唱された概念で，「コンカレントエンジニアリングとは，顧客の期待への対応を強調し，共同作業，委託，そしてチームの共有する価値をライフサイクルの全ての領域で，並行して進められる活動のインターバル全体でコンセンサスを導出するため，比較的詳細な情報の交換によって同期させられて意思決定が進行する方法で具体化させられた，統合された製品開発のためのシステマティックなアプローチである.」と定義されている．

サイマルテニアスエンジニアリングは，Winner [WPBS88] によって提唱された概念で，「統合された製品の同時並行な設計，生産，サポートを含めたそれらに関連したプロセスのための組織的なアプローチ．このアプローチは，開発者に初端の構想から廃棄までの製品ライフサイクルにおける全ての要素を考慮させることを起こすことが予想される．その要素とは，品質，コスト，スケジュール，そして顧客要求を含む.」と定義されている．

これらの考え方に基づいた木村 [Kim93] による設計生産活動モデルの提案，薮崎及び岩田ら [薮崎 93a] [薮崎 93b] [薮崎 93c] [薮崎 95] [岩田 95] [仲 義 94] による計算機支援システムの開発が行われている [MAE+95] .

これらのエンジニアリング方法論の定義は，全体としての大きな概念を示したものであり，具体的な方法論の提案は行われていないために，研究対象としては

- 設計グループの並行活動

- 設計活動と生産活動の同時並行活動

などの解釈の相違が生じている.

中間的なデータモデルによる製品モデルデータ交換

一方製品モデルの分野では，別々のアプリケーションでそれぞれの個別のモデルを用いる段階から，STEP[ISO95a] による統一的なモデル表現方法に基づいた，情報の交換と活用を計る段階に入っている.

STEP は EDI (Electronics Data Interchange) や SGML (Standard Generalized Markup Language) とともに，設計・生産・調達などの製品のライフサイクル中で関連するデータを計算機の支援の下で管理する CALS (Continuous Ackuisition and Life-cycle Support) [渡辺 95] と呼ばれる概念の中に組み込まれており，企業活動全般に渡る電子データ標準の体系化が進められていることになる.

また生産分野においても NC システムやロボットの発展はめざましく，設計生産における業務形態は，印刷された図面などを交換する形から，デジタル化されたデータを計算機システムとネットワークの支援の下で交換する形態へと変化しつつあると言える.

背景に関するまとめ

コンカレントエンジニアリングの提唱は，研究としては複数の解釈をもたらしてしまっているが，設計生産の活動中での設計活動と生産活動の並行化は，近年の生産リードタイムの縮小によるコスト低下に対する要求と，国際分業体制による生産の必要性から，実現せざるを得ない状況になっているが，その実行においては後述する解決すべき幾つかの問題点が生じている.

また，デジタルデータ化された製品モデルデータの現状に対して，Thomson [Tho94] から製品及びサービスを産み出す設計生産の活動そのものも何等かの形でモデル表現して取り扱い可能にし，より積極的に解析及び活用を行うことによる，結果として得られる製品及びサービスの質の向上と高効率化との達成が重要な課題として指摘されている.

次節では，本節で述べた背景に対して設計生産の活動を遂行する際に発生する問題点に関して述べる.

1.2 設計生産活動における問題点

1.2.1 設計生産活動の同時並行化における問題点

前節で述べた背景を踏まえて，設計生産における並行活動を進める際の本論文で解決すべき問題点について述べる.

図 1に本論文で対象と考える設計と生産の活動を並行して進める状況を示す．この図では，設計生産の活動は，設計と具体的に加工作業を行う生産グループ，それらの活動を管理する工程管理部門によって構成されている.

設計や生産における個別の活動は，計算機の支援によって効率化が進んでいるが，設計生産活動の計画，調整及び確認を行う管理業務は，依然として規則文書や伝達書類に従い管理担当者の手作業によって行われている．コンカレントエンジニアリングを適用した生産形態では，設計活動と生産準備，すなわち素材，工具や治具などの用意，が平行して行われるが，このような状況での生産管理に

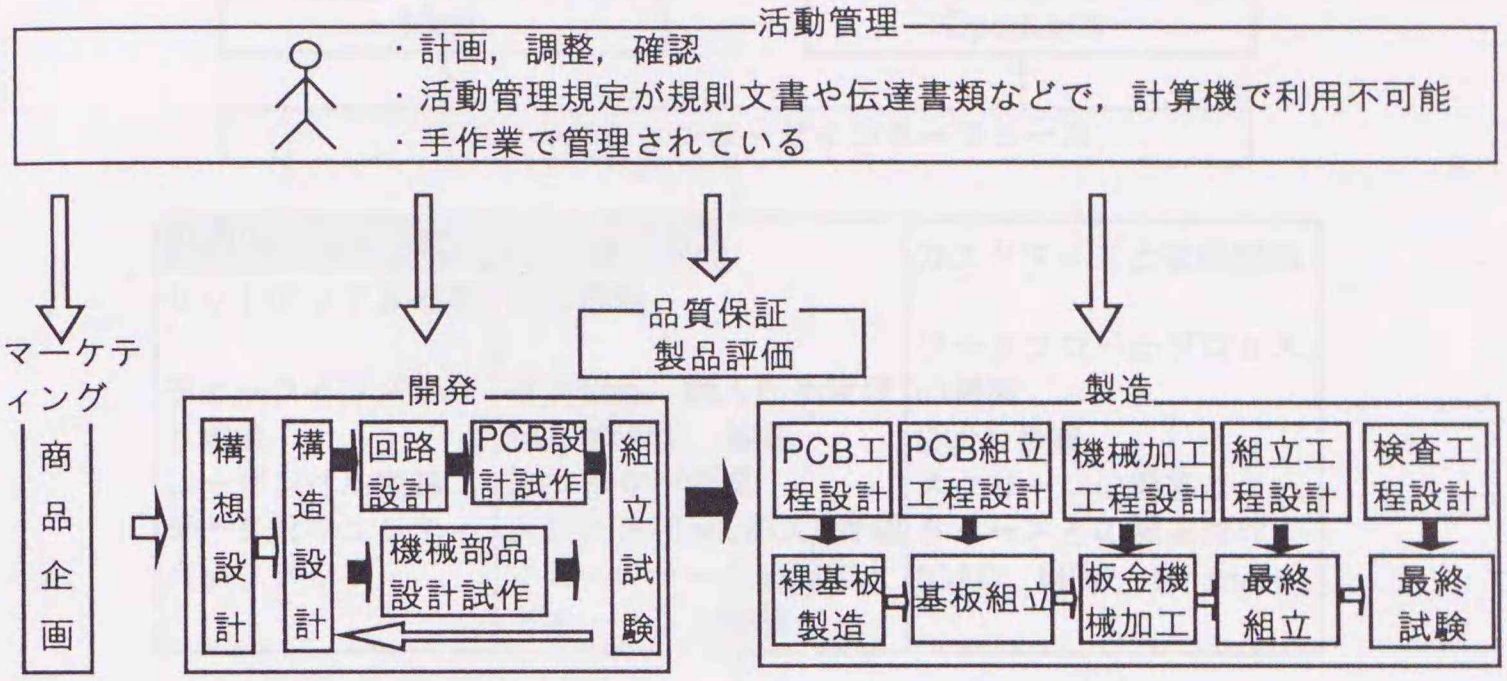


図 1: 本論文が対象とする設計生産活動の概要

は活動を単純に順列に並べて，前の活動が終わるのを待って次の活動を行っていた状況では考えられなかったような新たな問題が生じている．それらを次の項目で列挙する．

- 従来方式での管理業務は，地理的に分散されて行われる設計生産活動では困難になりつつある．
- 印刷物としてしか明文化されていない管理業務の規則と設計生産活動の成果であるデジタル化されたデータとの不整合のために，活動全体の効率を上げることができない．
- PL 法対策などのための，製品を保証するための情報として，製品の開発過程や製造過程に関する記録を計算機可読な形で保存することができない．
- 類似した設計生産活動で過去の事例の記録を自動的に検索することが困難となる．

これらの問題点の原因としては，設計生産における各種の活動に関する情報の取り扱い方に関連した問題と考えることができる．すなわち，設計生産の活動そのものに関する情報を記述表現し，計算機システムで応用する方法が確立されていない．

これらの問題を解決するための要求事項を整理すると次の項目になる．

- これまで手作業で行われて来た設計生産活動の管理業務を計算機支援システムの下で実行する形態を実現する必要がある．
- 上記システムを実現するために，以下の要件を満たす必要がある．
  - 要求仕様の決定から具体的な生産段階までの広い範囲を対象にして，個人個人が同時に複数の並行した活動を担当する状況での，設計生産の活動単位に発生する要求及び提案，データの参照及び変更，作業手順に関する規則と記録を表現するめに明確にモデルとして表現する必要がある．
  - 計算機やネットワークを用いた環境で利用可能な形で表現する必要がある．
  - それらのモデルの計算機システムへの実装を容易に行える必要がある．

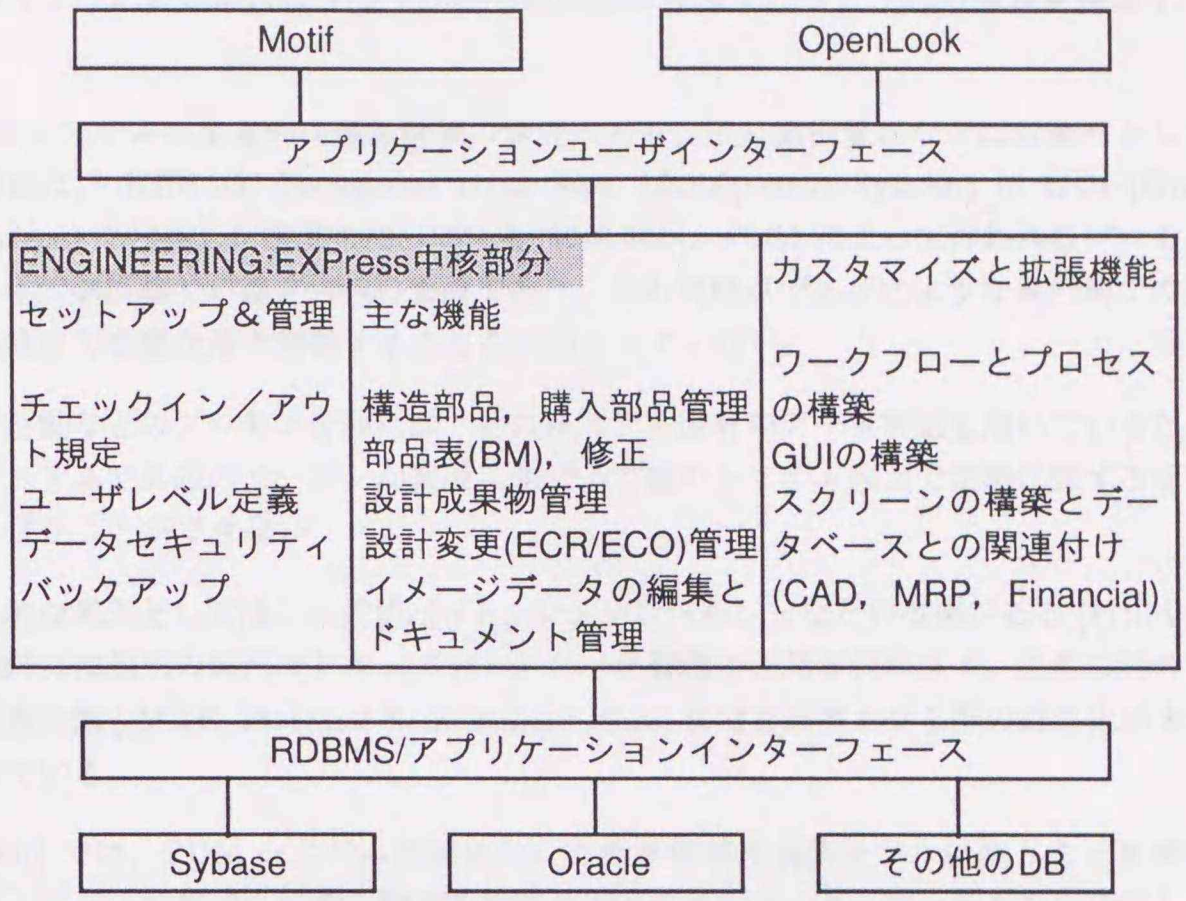


図 2: PDM の主な機能と構造

1.2.2 設計生産における既存の研究の限界

1.1節では，背景としてコンカレントエンジニアリングの提唱とデジタル化された製品データ交換に基づいた設計生産の形態の変化ついて述べ，1.2.1節では現実の製造における問題点について述べた．このような現状に対して現在設計生産の分野では大きく分けると，次の２種類の研究が対応している．

P D M (Product Data Management System)

PDM (Product Data Management System) は，概念設計から製造，保守管理のサポートに渡る製品の全ライフサイクル中において製品に関するすべての情報をコントロールすることを目的としたシステムである [平田 92] [木村 94]．管理する情報は，CAD ファイルだけでなく，ワープロ文書，DTP データ，表計算データ，ラスターデータ，解析データ，ＮＣプログラム，などの多様なデータを管理対象としている．

図 2 に [江沢 95a] から PDM の主な機能と構造を示す．
この文献では，PDM の機能を次の項目で整理している．

- 製品に関るすべての情報 : 部品情報，構成情報，ドキュメント，CAD ファイル，承認に関する情報などのような，製品を記述するのに必要な情報を管理する．
- 製品に関るすべてのプロセス : 承認や配布情報を含むプロセスとマネジメントの定義．製品のライフサイクル中での製品情報の管理を行う．

1.2.1節で上げた問題点に関するPDMでの対応の考察を行うと，次の項目を指摘することができる．

- PDMシステムの主目的は製品情報の管理にあり，製品原価管理などは対象外としている．その構造は， RDBMS (Relational Data Base Management System) に GUI (Graphic User Interface) を付加した構造を持っている [江沢 95b]．PDMによって行われるプロセスの管理とは，各活動段階での設計情報の管理であり，上の問題点であげたような部門間での変更の制限や記録までを陽な形で表現することを目的としていない．

- 設計活動などのプロセス管理には，そのシステム固有のデータ構造を用いているために，相互のシステムで共通のオープンな表現に基づいて他のシステム同士で活動に関する情報の交換などを行うことができない．

- 全社的な導入としては，エグザバイト，シュレンベルジェなどの事例がある [村山 94]．しかし，一般的には個別の部門でドキュメント／データ管理の運用事例が多く，生産工程の立案や資材所要量計画 (MRP: Material Requirement Plan) までを連携させる際の統合化が大きな課題となっている．

[江沢 95c] では，PDMシステム内部において業界標準や規格をフルに使うことを提案しているが，PDMシステム内部でも活動に関する情報やMRPとのインターフェースを明確にして生産工程立案や資材所要計画との連携を行うために，活動そのものに関するオープンな方法に基づく表現が必要であると考えられる．

S A D T (Structured Analysis and Design Technique) 手法

SADT (Structured Analysis and Design Technique) 手法 (State Carts モデリング技術とも呼ぶ) は，階層的かつ機能的な振る舞いとシステムの構造的な視点に基づく．システムを表現する方法論は図式的であり，現在は計算機にツールとして実装されている．この図式的仕様記述言語は，主に情報システムの設計と評価のために用いられた．SAS， SADT ，SARA，SREM などの計算機ツールがそのファミリである．

活動の内容に関する意味的な情報表現の分野では， IDEF[MMPd92][NISa][NISb] が草分け的な存在となっている．IDEF シリーズ (Integrated Computer-Aided Manufacturing (ICAM) DEFinition) は， SADT を用いたモデリング手法として， CIM 及びコンカレントエンジニアリングシステム構築を目的としている [MMPd92]．機能分析 (IDEF-0)，情報モデル構築 (IDEF-1)，データモデリング (IDEF-1X)，プロセス記述獲得 (Process Description Capture) (IDEF-3)，オブジェクト指向デザイン (IDEF-4)， 存在記述 (Ontology Description) (IDEF-5) ， 設計原理獲得 (Design Rathinale Capture) (IDEF-6) 等がある．

続いてフランスから GRAI[RZP89] と GIM の報告がなされ，またドイツにおいて [U.S83] の報告が行われた．これらも SADT 手法に基づくものである．さらに IEM[MSJ] におけるプロセスの概念も SADT に基づいている．

SADT 手法は，基本的にはプロセスに対する入力と出力を定義し，そのプロセスを入れ子構造で細分化するものである (図 3)． “ 入力 ” はその活動で処理される入力，“ 出力 ” は処理結果，“ コントロール ” は処理に関して参照される規定事項，“ メカニズム ” は処理を行う際に用いられる手法である．SADT 手法では，プロセスの細分化が単なる縦割りでしか行われず，同時に実行可能なプロ

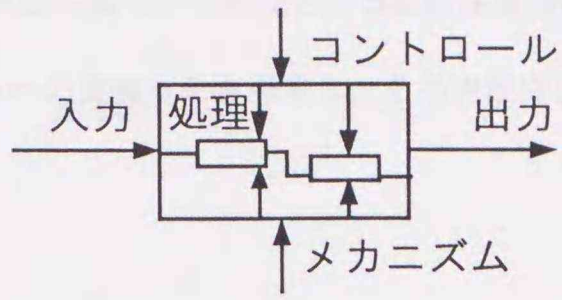


図 3: SADT 手法における基本構造

セスや，プロセスの前後関係なども表現できないため，並行したプロセスの表現が行われているとは言えない．また企業の組織的な境界線をモデルに写像してしまうこと，さらにモデル開発が機能の静的な分解の視点だけからしか行われない点が指摘されている [Ver94].

本論文における問題点のまとめ

1.2.1であげた課題の３項目

- 要求仕様の決定から具体的な生産段階までの広い範囲を対象にする必要がある．
- 計算機やネットワークを用いた環境で利用可能な形にする必要がある．
- 個人個人が同時に複数の並行した活動を担当する状況を対象にする必要がある．

に対して既存の研究では

- PDM では，活動に関する情報のオープンな表現
- SADT では，より動的な概念に基づいたプロセスのモデル化手法

がそれぞれ必要と考えられ，解決されているとは言えない．

上の２つの研究のほかにも，生産分野における活動表現に関する各種のモデルが存在するが，それらについては対象の分類に従って，第 3章，第 4章，第 5章において考察を行う．次節では，本節でまとめた問題点を解決するために本論文における目的と構成に関してより詳細に述べる．

1.3 本論文の目的・範囲・構成

1.3.1 目 的

1.1節で示した背景を踏まえ，1.2.1節で述べた問題点を解決して，設計生産活動の管理機能を支援する計算機支援システムを実現するために，本論文は以下のモデルの提案と計算機システム上での実装の検証を行うことを目的とする．

- 以下の構成からなり設計生産の活動を表現する設計生産コミュニケーションモデルを構築する．
  - 設計生産中に実行される各種の活動及び活動に関与する対象を統一的に表現できる基本モデル
  - 基本モデルに基づいた形式仕様記述言語を用いた記述方法

– 基本モデルと記述方法に基づいて表現される設計生産活動統合モデル

- モデルの計算機システムへの実装方法を提案し，その実現性を検証する．

1.3.2 範囲

報告を行う項目

前節で述べた目的のために

- 設計生産の活動に関する情報を蓄積／共有し，その内容に従う活動の質を保証するための基準を規定する．
- 設計生産の活動に関する情報を計算機内部で取り扱い，その意味的な妥当性を判定したり，直接計算機支援システムを実現するための方法論の確立．

上記２項目に対する解決法を提案し，その提案に基づいて研究全体としての方法論を確立し，実験的なシステムを計算機上に構築して最終的にその有効性を示すことを課題とする．

より具体的には本論文では，次に示す範囲での報告を行う．

- 設計生産の活動に関する情報を，人間と計算機とが取り扱える形で客観的に表現するための設計生産コミュニケーションモデルを構築する．即ち
  - 設計生産の活動を表現する形式仕様記述言語を用いて表現する基礎となるための基本モデルを確立する．
  - 形式仕様記述言語を開発する際の仕様を決める基準とするために，提案した基本モデルが動作する世界を完全に規定するために，提案した基本モデルをプロセス代数を用いて定式化する．
  - 設計と生産活動を表現することに特化した，形式仕様記述言語の開発を行う．
  - 大規模なモデルを構築する方法論を確立する．
  - 計算機による実装技術を確立する．
- 提案した表現技術に従って，設計部門から生産部門までの活動を表現する設計生産活動モデル設計生産活動統合モデルを構築する．
- 提案した形式仕様記述言語を実装した計算機支援システムの一般的な仕様を提案し，その実現のために重要と考えられるプロセス管理機能の実装を試み，シミュレータとして実現する．
- 基本モデルと形式仕様記述言語の機能に関して他の研究との比較を行う．

本論文での問題点への対処

1.2.1で示した問題点に対して，本論文では以下のような対応を取ることによって解決を行う．

- ある特定の活動に対応した活動プロセスと，その活動に対応する組織とを表現するために，その活動全体の管理者が，その活動に関連する作業者が集まった活動グループに相当するモデルを便宜的に定義する．これは，その活動が対象としている設計仕様データに対して組織内部で

の個別の作業者がどういった参照や変更などの権利を持っているか，権利情報や他の情報へのアクセスを制限する特権に関する情報を規定することによって，実際には個別の作業者が並行して複数の業務を遂行する状況を表現する．この組織モデルは，現実には存在していない場合や，物理的に離れた作業者であっても，表現することができる．

- 設計生産における活動を表現することを主目的とした形式仕様記述言語を提案することで，例えば設計部門と生産部門の活動の同期を取るために，どういった状況の進展 (イベントの発生) によって，次に行うことができる活動が変化するか，どういった情報の伝達や設計データの変更が可能であるか，を部門の管理者によって明確に規定し，計算機支援システムでその内容を実装することが可能になる．

- ある要求仕様に対して，どの設計仕様の規定を行ったかを，設計活動（プロセス）において，やりとりされた要求仕様の提示と設計仕様の提示の連続として記録し，本論文で提案した形式仕様記述言語で記述表現する．また，製品を生産する際においても，その時点でどういった製品モデルデータが，設計生産の活動の中で用いられていたかを記録するために，各設計活動，生産活動間で，どのような情報のやり取りが行われていたかを，記述表現する．

また次のような効果も考えられる．

- ある活動（プロセス）中において設計仕様変更のイベントが発生した場合に，その情報をどの部門（対象）に伝達するかを，その活動を管理する人間が規定して，プロセス内部での情報の配分先として，明確に規定することが可能になる．

- 本論文で提案する基本モデルでは，活動に関与する対象を内部活動を表現しない入出力対象として扱う，このことによって，資材所要量計画部門も，その部門で受け付けることができる入力（インターフェース）として定義し，その間の連携活動は，相互の部門間での情報交換を表現するプロセスとしてモデル化することができる．

1.3.3 構 成

上で示した方針に従って 2章以降では，次の構成に従って報告を行う．本論文の構成を図 4に示す．

2章 各章毎の個別の問題点の詳細な分析と本論文における方針の確立

3章 設計生産過程を形式仕様記述言語を用いて表現するため基本モデルの提案，提案した基本モデルを実現し，形式仕様記述言語でのモデル表現機能を実現するための，基本モデル内部でのメッセージの交換やプロセスの生成や停止などを実現するプロセス管理機能の定式化

4章 基本モデルに基づき設計生産活動を表現するための形式仕様記述言語の開発

5章 設計生産の種々の活動を一つの枠組みで表現するためのフレームワークの提案，及び設計生産に関する設計グループの活動と生産活動における生産設備の活動を表現するモデルの構築，他の方法論との比較

6章 形式仕様記述言語を用いて表現された設計生産過程に関する情報を実装するシステムの一般形の提案と，具体的な計算機アプリケーションとしての実現するためのプロセス管理機能の実装の検証

7章 全体を通じての結論

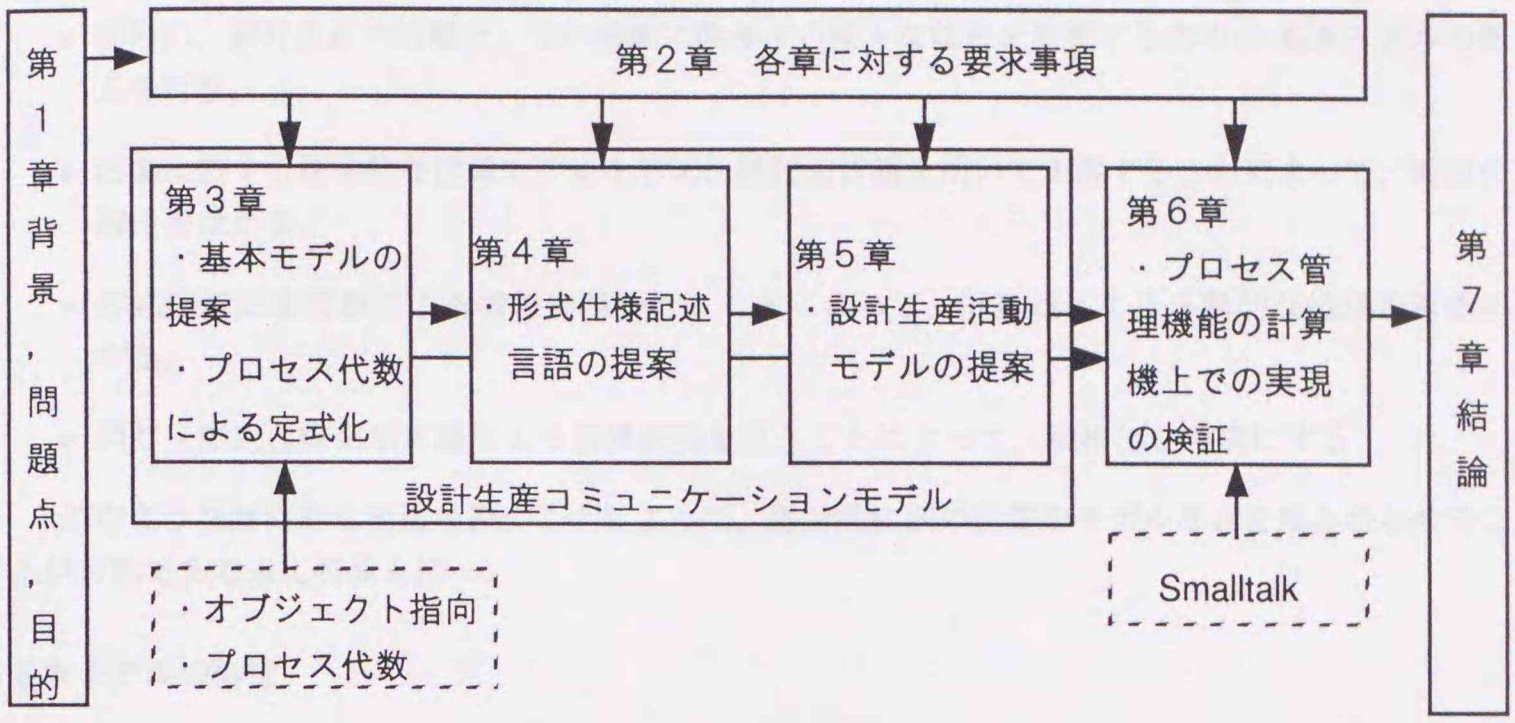


図 4: 本論文の構成

1.4　本論文の特徴

本節では，研究対象，方法論，基本モデル，形式仕様記述言語，設計生産活動統合モデル，拡張性，他の計算機システム／工作機械命令／ロボット操作言語との関連に関する特徴，計算機支援システムの特徴，の各項目に関して本論文の特徴をまとめる．ここでは，2章以降で述べる内容にも触れて本論文における全体的な特徴を述べる．

研究対象の特徴

- 設計生産の過程で扱われる製品に関するデータを直接的な対象とせずに，その活動を情報の交換の観点から表現するモデルを対象としている．
- 概念的な設計段階から実際の生産，保守及び廃棄などの製品のライフサイクルの中で行われる様々な活動を対象としている．
- 上記の活動の中でも同時に並行して実施される活動の表現を対象としている．
- 上記の対象の活動の中で，実際に活動に関与する，設計者，設計者のグループなどの作業者及びその組織， CAx システム，データベース，などのアプリケーションシステム，入出力されるデータ，そのデータモデルの構造に関する情報，ロボット，工作機械，搬送設備などの生産設備，などを表現することを対象としている．

方法論の特徴

設計生産の活動に関して，単純に個別のモデルを提案することに注目せずに，初めにそのモデルを形式仕様記述言語を用いて表現する方法論を確立し，設計から生産に至る設計生産活動モデルの表現を行い，計算機による実装を行う階層的な方法をとっている．具体的には次の手順に従って行う．

- 初めに，設計生産の活動と，その活動に関与する様々な対象を表現するための 基本モデルの確立を行う．
- 活動に関する意味的な情報モデルを形式仕様記述言語を用いて表現することによって，問題の解決をはかる．
- 形式仕様記述言語による情報表現を行うことによって，計算機による自動的な処理を可能にする．
- 同じく形式仕様記述言語による情報表現を行うことによって，規格化を可能にする．

このような階層的な構造を持つことによって，部分的に別の性質のモデルなどを組み合わせることが可能である点に特徴を持つ．

基本モデルの特徴

本論文では，プロセス代数とオブジェクト指向に基づいた基本モデルを提案する．このために，これら２つの概念による次の利点を得ることができた．

- 設計生産の過程を，その過程に関与する様々な対象の定義と，その間に存在するプロセスとの関係として表現する．
- 設計生産の活動に関与する多種の対象の間で交換される情報を伝達する媒体として，プロセスをモデル化する．
- 設計生産の活動に関与する様々な対象は，内部の活動には注目しないで上記のプロセスから見た情報の入出力対象 (インターフェース) として表現する．こうすることによって，設計時における具体的な人間の思考過程のような，現在のモデルでは表現不可能な活動は表現せずにブラックボックスとして扱うことが可能になる．このことによって表現不可能な対象モデルでも組み入れて，広範囲のモデルとして構築することが可能になる．
- 上記の入出力対象からプロセスを経由することによって行われる情報の入出力の観測を，そのプロセスの活動の記録として扱う．
- 活動ごとに複数のプロセスが存在し，その活動に関する情報は当該プロセスを経由することで，複数の参加者が複数の並行した活動を実行している状況を表現することができる．
- この基本モデルの対象外のモデルもブラックボックスである入出力の対象として表現することで，複数の概念に基づいたモデルも組み合わせることができる．

形式仕様記述言語の特徴

本論文で提案する形式仕様記述言語は，上で述べた基本モデルに基づいた形で，設計から生産にいたる活動を表現することを主目的に定義した．従って，基本モデルにおける特徴に加えて，具体的な活動表現を容易にする次の特徴を持っている．

- 設計や生産における活動をプロセスとして，それ以外の設計から生産まで関与する多様な対象を入出力のインターフェースとして表現する．

- 情報を入出力する対象とプロセスの区別を行わないで，同じ入出力対象である一種のプロセスとして扱う．
- プロセス及び入出力対象は，そのプロセス (入出力対象) に対して起こる情報入出力のイベントと，そのイベントへの対応として表現する．これは，そのプロセス又は入出力対象が受け付けることのできるメッセージの集合としての意味を持つ．
- 基本的なプロセスの定義は，プロセスの名称，起動される時に渡される引数，内部変数，入力されるメッセージ，及びそのメッセージに対する反応によって定義される．
- プロセスの動作は，起動時の初期化の後で無限ループ中でメッセージが入力されるのを待ち，メッセージの入力があった場合に予め規定された反応を取るイベントドリブン形式でモデル化される．
- 形式仕様記述言語には，別のモデルで定義された内容を継承する機能が用意されている．これによってメッセージと対する反応からなる特定の用途向けのインターフェース仕様を定義しておき，その内容を多重継承することによって設計生産活動モデルにおける入出力仕様を定義する．
- 入出力対象からは，プロセスを経由してしか情報を入力できない形でモデル化を行う．

設計生産活動統合モデルの特徴

- 特定の目的に応じた入出力対象とプロセスからなるリソースを定義し継承を行うことによって，設計生産活動モデルの構築を行うことができる．
- 各リソースは，特定の入出力対象と関係する活動において当該入出力対象に対してサービスを提供するプロセスの組合せとして定義する．
- 抽象化されたファイルシステムやデータベースに相当する入出力対象，データモデルの構造に相当する入出力対象を定義することで，モデル構造の定義，モデルデータの入力などの活動も表現することができる．
- 活動に関する全ての情報をプロセスを経由させることによって，特定のプロセスを時系列的に通過する情報の流れとして複数の対象が関与する複雑な活動を表現できる．
- 仮想的な活動グループが，情報の交換及び分配を制御することによって，グループ組織内の情報の閲覧，操作，承認などの権限を表現することができる．
- 一種類のプロセスを定義しておき，そのプロセスをスケジュールに従って再帰的に起動させることによって次の特徴が得られる．
  - スケジュールデータと，その各活動におけるプロセスのイベント定義を分離することができる．
  - グループによって行われている活動で，そのグループの構成が活動の進展に応じて動的に変化する状況を表現することができる．又グループの構造変更を行う権利なども上記と同等に定義することができる．
  - プロセスの並行動作の可能性や，活動の終了に関する依存性，などをアルゴリズムの定義とは別に，スケジュールデータの中で表現することができる．

  - 特定の活動におけるスケジュールとプロトコルを分離することによって，プロセスの表現を，活動ごとに入出力される情報と，その情報の入力をイベントとしての対応とに明確化することができる．

- 生産に用いられる部品などの移送や加工なども，情報の入出力対象として表現することで，本論文における基本モデルの範中で表現することが可能になる．

拡張性の特徴

- 単に形式仕様記述言語を用いた表現技法を確立するだけでなく，設計生産活動モデルを構築する方法論を合わせて確立するので，同じ基本モデルに従うモデルであれば，異なる分野の活動モデルでも縦横に組み合わせて開発を行うことができる．
- 本論文で提案する設計生産の活動は，同じ基本構造に基づいて，それぞれの分野ごとのプロセスとインタフェースとを定義することでモデル化される．これらのモデルの各プロセスは，ともに並行して相互に連携して活動できる．
- 別個の目的のプロセスやインタフェース仕様でも，それらを継承させることで，統合することができる．
- 基本的な表現技法を確立した上で，モデルを構築して表現するため，一度表現の実装の方法論が確立されると，表現された様々な活動モデルを同じ方法論に従って実装することが可能になる．

他の計算機システム／工作機械命令／ロボット操作言語との関連に関する特徴

- 一般の CAD システムなどで用いられている操作マクロなどを，そのシステムに対する入力プロセスの定義として考えることによって本論文で表現できるモデルとして扱うことができる．また特定の形状要素に対する操作の依存性を表現することで，その形状要素での属性の依存性を表現して，結果としてパラメトリックモデルの一種として定義することが可能となる．
- 工作機械に対する CL データ (Cutter Location Data) も，その対象である工作機械 (のコントローラ) に対して連続的に一連のデータを送り込む一種のプロセスと見ることができる．例えば，複数の工作機械に対して異なったタイミングで命令を送り出すような状況も，上記の考え方で，JEfsdl で表現したプロセスが時間的な制限を受けて，並行してデータを出力するモデルとして表現することができる．
- ロボット対する移動命令も，その対象であるロボット (のコントローラ) に対して連続的に一連のデータを送り込む一種のプロセスと見ることができる．
- 設計過程で設計者と CAD システムの間で入出力されるデータや，エンジニアリングデータベースで保存されるデータ，そのデータの意味的な情報までも入出力対象として扱うことができる．例えばSTEP における物理ファイルの構造定義などもデータへの入出力過程として写像操作を表現できる．
- 上に述べた，CAD システムのマクロ，工作機械に対するＣＬデータ，ロボットに対する移動命令も，単純に連続的なデータを送り出すプロセスとして見るのならば，本論文で提案した形式仕様記述言語で表現することが可能であり，システムごとの書式変換を行うことによって，その目的固有のフォーマットに変換することが可能であると考えられる．

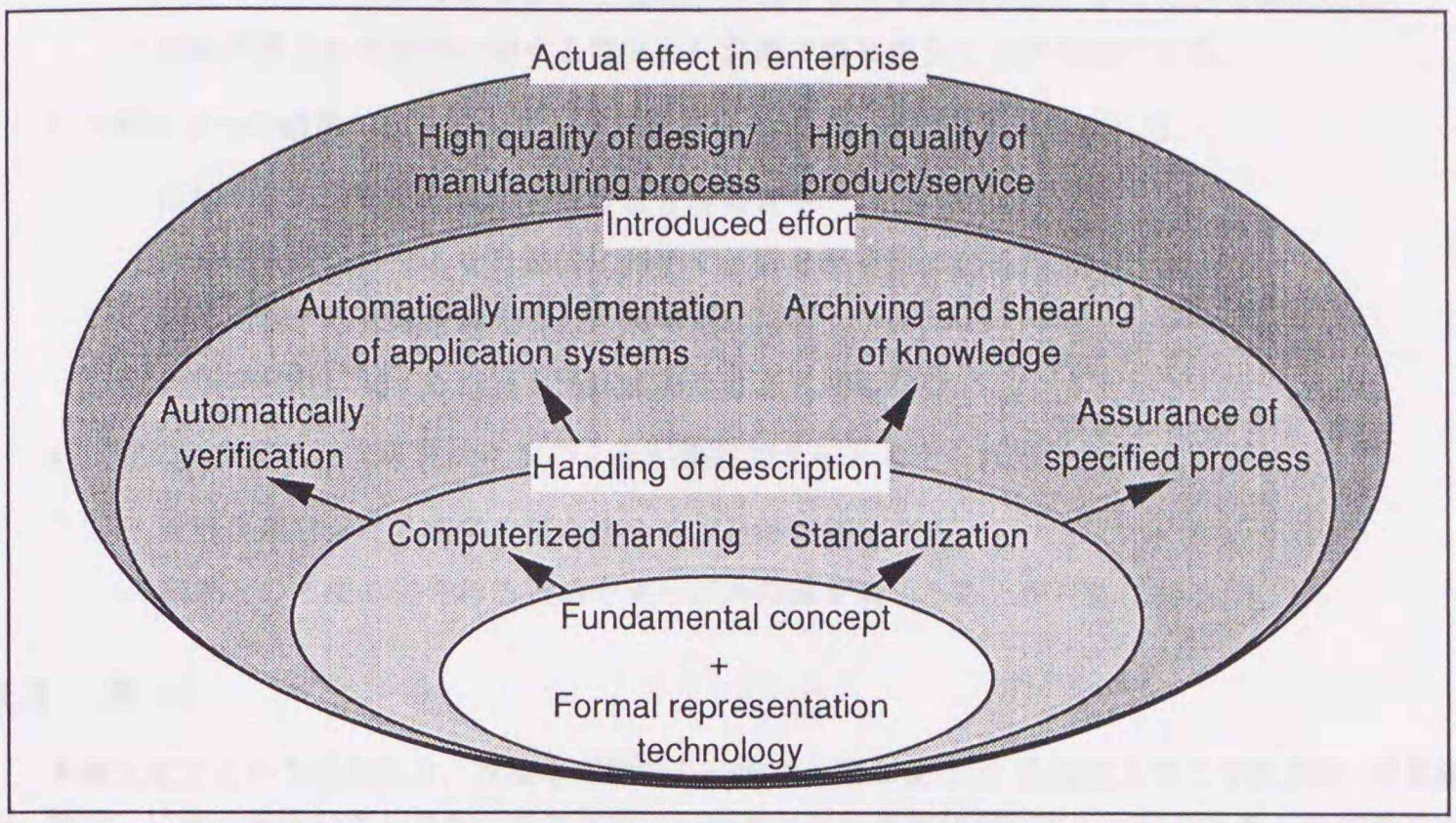


図 5: 提案された方法論の摘要によって期待される効果

プロセス管理機能の実装による特徴

- システムの構成をモデュールに分けて提案している．このことによって次の特徴を得ることができる．

  1. 各モデュールの機能を明確にしている．
  2. 各モデュールでの他のアプリなどに対する依存性を，モデュール毎に限定することができる．

- 単なるシミュレーションとして実現するだけでなく，種々の実装が行えるように考慮した形になっており，ブラウザとは独立したクラスとして定義しているために具体的なアプリケーションに依存していない．実際の計算機支援システムでは，このクラスライブラリは，必要に応じて他の C++などのクラス階層へ変換して実装することができる．

- 活動表現は専用の形式仕様記述言語を用い，その活動を計算機環境内で実現する場合には計算機言語に変換して用いる方式を取っている．そのために，シミュレータの開発などは，プロセスの操作を行い易くまたプロトタイプが容易な Smalltalk を用いた形で行うことができる．

期待される将来的な効果

本論文における方法論を摘要することによって期待される将来的な効果を図 5に示す．

- 提案した基本モデルと形式仕様記述言語とを用いることによって，活動に関する情報を扱う次の利点が生まれる．

- 提案した表現技術による設計生産活動に関する情報を規格化することが可能になる.
- 形式表現された活動に関する情報を計算機で取り扱うことが可能になる.

- 上記の２つの情報の取り扱いにおける利点によって，次の効果が得られる.
  - 活動内容の自動的な判定を行うことができる.
  - 計算機支援システムを自動的に開発することが可能になる.
  - 活動に関する知識を蓄えることが可能になる.
  - 特定の活動に関する内容を保証することが可能になる.

- 上記の効果によって最終的に次の２点が達成できると考えられる.
  - 設計生産における活動の質を高めることが可能になる.
  - 結果として生み出される製品やサービスの質を高めることが可能になる.

1.5 謝 辞

本論文をまとめるにあたり，貴重なご指導と御鞭撻を賜りました 北海道大学工学部教授 岸浪建史 博士 に深く感謝いたします．また本論文の審査と共に貴重な御助言を多数頂きました北海道大学工学部教授 五十嵐 悟 博士，北海道大学工学部教授 島 公脩 博士，北海道大学工学部教授 嘉数 侑昇 博士に心から感謝いたします．また本論文に対して深い御理解と一貫して有益な御助言を頂いた 北海道大学工学部助教授 金井 理 博士 に感謝いたします．

本論文は，1991 年秋から 1992 年春まで北海道大学工学部 精密工学科 精密機器学第二講座にて行われたプロセスゼミでの検討結果を元にして行われたものであります．このゼミナールに参加し，数々の助言をいただいた北海道学工学部助手 田中文基 氏を初め，当時のプロセスゼミメンバーの御協力に感謝いたします．

また，本論文を投稿論文としてまとめるにあたり，適確なご指導をいただきました北海道大学工学部教授 斎藤 勝政 博士，大阪大学工学部教授 三好 隆史 博士，北海道大学工学部技官 渋川 勝久 博士，北海道大学工学部技官 近藤 司 博士，北海道大学工学部助手 楢原 弘之 博士らの御協力に感謝いたします．

さらに幾度かのワークショップにおいて，研究の方針と論文の構成に関して極めて貴重な御助言を頂いた，工業技術院 機械技術研究所 首席研究員 小島俊雄 博士，東京理科大学助教授 榎本 進 博士の御二人に心から感謝いたします．

また研究全般に渡って助言をいただいた，旭川工業高等専門学校 三井 聡 氏と函館工業高等専門学校 山田 誠 氏に感謝いたします．

第 4章における機械設計活動における設計変更過程の作業情報の使用を許諾していただいた株式会社日鋼デザイン 山田 富士夫氏の御協力に感謝いたします．

本論文の第 5章における設計グループ活動モデルの開発に協力していただいた，当時の北海学園大学学生 須藤和茂，坂橋徳彦の両君に，また第 6章におけるシミュレータの開発に参加してくれた当時の北海学園大学学生 永瀬善章君の協力に感謝いたします．

また本論文の印刷に関して御助力を頂きました，大阪大学工学部講師 高谷 裕浩 博士に深くお礼申し上げます．

最後に，本論文の作成にあたり多くの御協力を頂いた，勤務先の北海学園大学工学部関係者及び電子情報工学科の各教員の方々に心より感謝いたします．

1.6 本論文で用いる主な記号

本論文で用いる主な記号に関して予めまとめる．この記号は，主に Milner [Mil89] によるプロセス代数 CCS (Calucuas of Communicating Systems) で用いているものであり，本論文でも CCS に準じた記号表現を用いている．このため以下の表 1〜5で，記号とその名称について示す．

要素		
要素集合	要素名	要素の種類
$\mathcal{A}$	$a, b, \ldots$	名前
$\overline{\mathcal{A}}$	$\overline{a}, \overline{b}, \ldots$	相補名
$\mathcal{L}$	l	ラベル
Act	$A,B,\ldots$	アクション
$Act*$	$r,s,\ldots$	アクションシーケンス

表 1: 本論文で用いる主な記号 (1)

集合	
要素名	要素の種類
0	空集合
$\{z : \Phi\}$	zの集合，即ちΦ
$\tilde{z}$	順番付けられた集合 $\{z_i : i \in I\}$
$< v_1,\ldots,v_k >$	値の並び

表 2: 本論文で用いる主な記号 (2)

アクション	
要素名	要素の種類
τ	静的ないし内部アクション
$\overline{l}$	ラベルコンプリメント
$\hat{s}$	τが除去されたアクションシーケンス s
r/s	s 上での rの ??
1	単位アクション
$\alpha \times \beta$	アクションの積

表 3: 本論文で用いる主な記号 (3)

基本エージェント

要素名	要素の種類
$\alpha.E$	プレフィックス
O	動作しないエージェント
$E+F$	和
$\sum_{i \in I} E_i$	添え字つき集合上での和
$E\|F$	積
$\prod_{i \in I} E_i$	添え字つき集合上での積
$E \setminus F$	積
$E[f]$	リラベリング
$\{\tilde{E} \setminus \tilde{X}\}$	並列置換
$\mathbf{fix}(\tilde{X} = \tilde{E})$	再帰

表 4: 本論文で用いる主な記号 (4)

値付きエージェント

要素名	要素の種類
$\alpha(x).E$	プレフィックス (値の入力)
$\overline{\alpha}(x).E$	プレフィックス (値の出力)
if b **then** E	条件
$A(\tilde{x}) \stackrel{def}{=} E$	添え字つき集合上での和
$\hat{E}$	基本式への遷移

表 5: 本論文で用いる主な記号 (5)

2　設計生産活動における問題点の詳細化と解決の方法論

設計生産の活動表現と計算機支援システム実現するには，一つの活動モデルを単純に提案するだけでは不完全であり，広範囲の設計生産の活動に関与する各対象や，それらによって実行される活動に関する基本モデル，その基本モデルに基づいた表現技法，その表現技法に基づいた基本的なモデル構造，計算機による実装の方針など，多々の問題点がある．

そこで本章では，1.2.1での問題点に対して，1.3.2で示した構成に従って個別の項目に分類整理し，本論文で解決すべき課題から個別の問題点への詳細化を行う．

2.1　設計生産活動と活動に関与する対象とを表現する基本モデル

複雑な設計生産活動そのものを独立した形で扱う表現能力

設計生産の活動表現を課題として考えた場合には，次の各種活動までもを表現の対象として考える必要がある．

- 設計活動
- 承認活動
- 連絡／交渉活動
- 搬送活動
- 加工作業

上に列挙した活動の表現を考えた場合には，これらの各種活動の内容を個別に表現することも必要であるが，それだけではなく活動を単独に表現し，なおかつそれらの相互の関係を表現することが重要と考えられる．すなわち，

- 複雑な設計生産活動そのものを独立したプロセスとして扱う表現能力
  - 個別の作業者が複数の作業に関与する活動状況の表現．
  - 複数の作業者が並行して同時に実行される各種活動の表現．
  - 活動相互の関係

プログラム言語に代表される従来の手続き的な表現技法は，一連の手続きを構造的に考えるための枠組みを提供するが，基本的には唯一のプロセスが実行されている状況にしか注視していない．このため，上に示した複数の活動が同時に実行される状況の表現には，不適当と考えられる．その理由として，各活動が動作している際の活動同士の動作の時間的な前後関係，特定の動作が実行される以前に完了していなくてはならない活動，同時に動作することが可能な活動の種類，などに関する情報が表現できないからである．このため，設計生産において実行される各種の活動を明確に表現するための，活動あるいはプロセスが如何なる定義によって表現されるか，に関する基本モデルを本論文では確立する必要がある．

図 6に，具体的な並行活動の概念図を示す．この図では，実際の設計生産活動では一人の設計者が複数の設計グループに所属し，それらが同時に並行して関与する状況を示している．具体的には，一人の設計者から複数のグループへ何等かの形で，同じ活動を行ったり同じ情報を同報するなどの活動である．

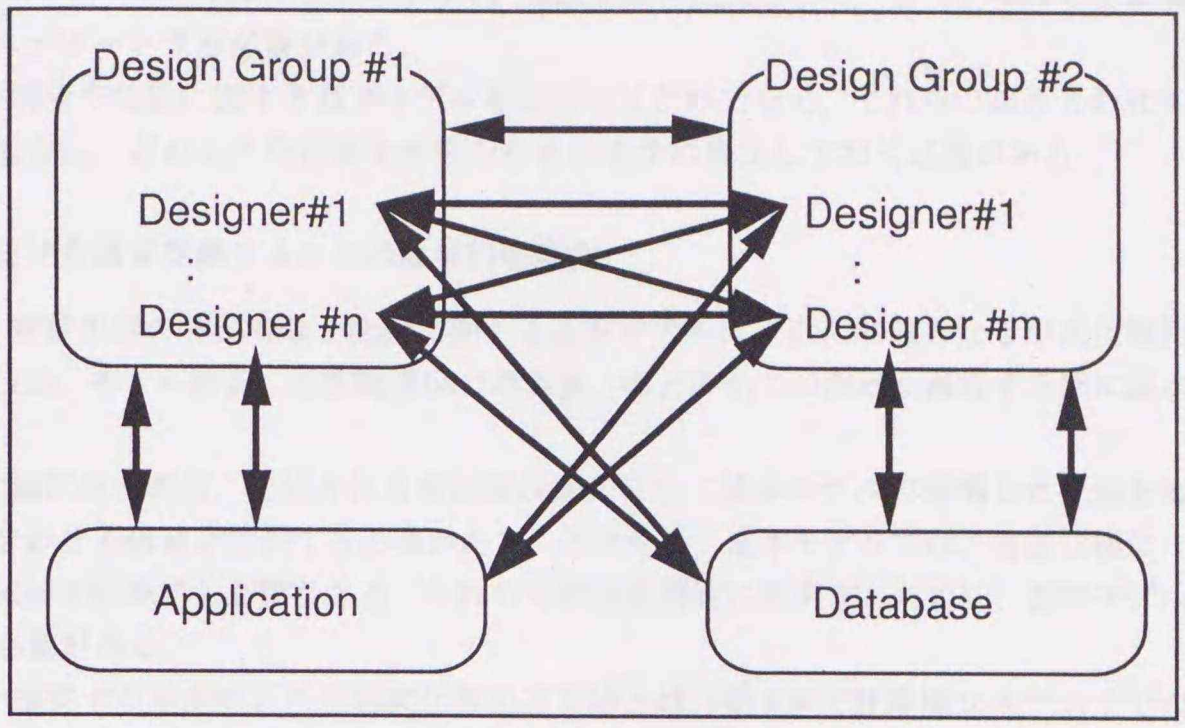


図 6: 並行活動の概念

設計生産の活動に関与する多様な対象を扱う表現能力

また図 6では，アプリケーションとデータベースとしか示していないが，実際の設計生産の活動に関与し，相互に協調しあう対象として次の項目が考えられる．

- 各設計者や設計グループなどの人的資源
- CAD，エンジニアリングデータベースなどのアプリケーションプログラム
- 複数の工作機械やロボットなどの製造設備
- 製造設備や貯蔵設備などで操作の対象となる製品，素材及び工具などの部品類
- 入出力されるデータ
- データの構造に関するデータ (メタデータ)

ここでのデータの構造に関するデータとは， STEP [ISO95a] における EXPRESS [ISO95b] で記述表現された製品データモデルの構造や，データを格納する物理ファイル [ISO95c] などが挙げられる．例えば，データモデルを構築するプロセスや， STEP データモデルから物理ファイルフォーマットへの写像などをプロセス表現の対象として考えた場合には，このような情報も操作の対象とする必要があると考えられる．

設計者の設計活動を表現する問題を考えた場合，その思考の過程までも含むような非常に広範囲の対象が考えられる．従って設計生産活動の表現では，設計生産の過程に関する概念だけでなく，このような設計生産に関する広い範囲の対象を，曖昧さ無く明確に表現する必要がある．

さらに，上で挙げた対象は，活動中にその構造が変化することが考えられる．例えば，設計グループの各要員は，特定のグループに属して設計活動を行うと考えられる．この設計グループの構成は常

に一定ではなく，設計活動の進展に応じて，臨機応変に変更されるべきで，それらを表現するモデルももちろんサポートする必要がある．

また，対象や活動に関する基本モデルを確立するだけでなく，それらの組み合わせて相互作用を行わせた場合に，どのような意味を表現できるかを予め規定しておく必要がある．

形式仕様記述言語を定義するための数学的明晰性

この，設計生産の活動対象と活動に関する基本モデルは，次に問題となる形式仕様記述言語を用いた表現技法，モデル構造，計算機環境での実装，などの全ての問題を解決する際に基本となる問題である．

形式仕様記述言語は，記述される最低限の意味として基本モデルで提唱した機能を実現し，それらを組み合わせる構造を提供する必要がある．そのため，基本モデルでは，言語仕様に一対一に対応する形で機能を表現する必要がある．それらの機能を明確に表現するために，数学モデルによる定式化を行う必要がある．

また，提案する基本モデルと形式仕様記述言語とは，第6章で計算機システム上での実現可能性に関する検証を行う必要があるが，基本モデルや形式仕様記述言語が完全に動作するためにシステム内部で満たしいる必要がある一貫性の管理 (プロセス管理機能) を明確に定義しておく必要がある．

2.2 設計生産活動を表現する形式仕様記述言語

前節 2.1では，設計生産の活動を表現するための方法論として，各種表現の原始的な要素となる基本モデルと，それらを組み合わせた際に表現できる意味とを規定することの重要性について述べた．さらに，大規模な設計生産活動のモデルを構築して表現するためには，モデルの構造化や一貫性の維持，計算機によるモデル化支援などを行う必要が考えられる．

その解決にはSTEP で用いられている形式仕様記述言語を用いた表現技術が有効な手段になると考えられる．意味モデルの統合化及び規格化の進む製品モデルの現状に対比して，設計生産活動に関する情報表現では，逆に上の項目が具体的な問題点として考えられることになる．即ち次のような現状があると考えられる．

- ある基準に基づいて特定の設計生産の活動の質を保証することが出来ない．
- 設計生産の活動に関する知識の蓄積／共有が行われていない．
- ある設計生産の活動が妥当であるかを判定するための，基準と判定方法が確立されていない．
- 設計生産活動に関する意味的な情報を計算機処理することが行われていない．
- 設計生産活動に関するアプリケーション開発を推進するための基盤が確立されていない．

二木 [二木 92] によると，このような活動の記述表現は，それに基づいて規格化やシステム開発を行うための基準となるべきものであり，次のような性質をもつことが要求される．

無矛盾性 規定されている内容が一貫しており，自己矛盾がないこと．

完全性 必要な内容がすべて規定されており，特殊な知識などを背景として持つ必要のないこと．

非曖昧性 規定されている内容が，複数の解釈を生まずに一意に特定できること．

明晰性 規定されるべき内容が理解しやすく記述されていること.

上の３つの特性を得るための仕様の記述方法として，次の２つの方法論がある.

- 自然言語による仕様表現
- 形式仕様記述言語による表現

自然言語を用いて対象を述べる仕様表現は，最も基本的な方法である．しかし自然言語による仕様記述では，意味を定める厳密な規則が利用可能な形で提供されない．また，その仕様の作成者の常識やその仕様の背景となっている技術に関する知識などに依存した記述が暗黙哩に行われる可能性が大きい．従って，無矛盾性などの性質を判定する際の基準となるべき「仕様の意味」を客観的に特定することが難しい．従って自然言語でかかれた複雑なシステムの仕様が，無矛盾性，完全性，非曖昧性などの性質を満たすかどうかを客観的に判定することは極めて困難になる．そのため，自然言語を用いてモデルを記述するだけでは不十分であると考えられる.

モデルを自然言語で表現する方法に対して，形式仕様記述言語を用いて仕様を記述する方法がある．この方法では，通常は形式仕様記述言語 (Formal Specification Description Language) を用いる．形式仕様記述言語では，利用可能な基本的な言語要素と，その要素が持つ意味があらかじめ与えられ，言語要素を組み合わせる際の並び方と，その組み合わせによって得られる意味が構文規則によって厳密に定義することが可能になる．これらの技術は一般的には FDT (Formal Description Techinique) と呼称されている [二木 90].

形式仕様記述言語用いて表現を行うには，次のような規則を予め定義する必要がある [二木 92].

構文規則 どのような記号の並びが仕様文書であるかを定める構文規則．通常は形式文法 (生成文法) で与える.

意味モデルと意味規則 仕様の意味モデル

構造化方法 何を単位にして仕様をモデュール化し，それらのモデュールをどのように組み合わせて仕様を構成するかを定めた規則.

形式仕様記述言語を仕様の記述に用いる際の利点として，以下の項目が報告されている [二木 92].

1. 無矛盾性，完全性，非曖昧性などの重要な性質を機械的に判断できる可能性がある.
2. 仕様 (情報) 自体の機械的処理が可能であり，仕様の解析，蓄積，再利用などに計算機を有効に活用できる可能性が高い.
3. 仕様のモジュール化が行いやすい.

これらの形式仕様記述言語は，現在次の分野で広く用いられている [水野 94].

- 形式仕様記述言語
  - S D L (Specification and Description Language) [若原 90] [CCI88]
  - Estele (Extended State Transition Language) [岡田 90]
  - LOTOS (Language of Temporal Orderling Specification) [高橋 90a] [ISO89]
  - ペトリネット [椎塚 92] [W.88] [奥川 95]

- Z[PST95]

- プログラミング言語
  - CHILL[高橋 90b]
  - C [KR98]
- プロトコル記法
  - ASN.1 [森野 94][高橋 90b]
  - TTCN[勝山 90]
- 製品仕様記述
  - VHDL [Jame89]
- 製品モデル
  - EXPRESS [木村 95] [Fow95]

本論文で目的として上げた設計生産活動の表現と活用を行うためには，形式仕様記述言語による表現方法の開発が必要であるが，その中でも

- 基本モデルとして提案した要素の表現
  - 設計生産における活動
  - 設計生産に関与する対象
  - 交換される情報の表現
- モデル化を容易にする構文仕様
  - 単純なシグナルの送受信ではなく，より高度な情報の交換
  - 定義内容の継承機能
  - 情報の更新の構造が変化できるメカニズム
  - 必要最小限の言語要素による表現

などが必要であり，この方式に基づいたモデル化と実装を行うことが必要と考えられる．

2.3 広範な活動を包含する設計生産活動統合モデル

広範な対象領域の活動表現を効率行うための構造

2.2では，設計生産の活動を表現する基本モデルと形式仕様記述言語の重要性に関して述べた．本論文では，設計生産における広範囲な領域での活動表現を目的としている．従って具体的な活動モデルとして，端的には設計活動のグループモデルから生産設備に対するデータをダウンロードするまでの活動をそれぞれ表現し，またそれらが相互に関連するモデルを構築する必要がある．

大規模なモデル構築は，同様な大規模プログラムの開発と相似性を持っている．一つのルーチンだけからなる大規模なプログラムの開発を行う場合，プログラム開発に必要とされる人間の数が多く

なり，開発要員相互の連絡に必要な時間が増加して開発効率が低下してしまう [Davi93]．また，部分ごとの規格化／部品化が行えないと，全体が完成するまでに個々の動作の試験を行うことが出来ないために，プログラム全体としての安定性や整合性に欠けてしまう恐れがある．このために，大規模なプログラムを個々の人間が独立して開発できる小さな構成要素に分解して開発し，その後組み合わせてプログラムを完成させる方法論が採られている [Davi93]．大規模なモデルの構築でも同様に，全体を小さな構成要素に分解し，それらを組み合わせてモデルを構築できるようにする必要がある．

しかしモデル開発の場合には，複数のモデルを組み合わせる際に，各モデルの機能が重複したり欠けてしまい，小さなモデルを組み合わせてモデルを構築するだけでは不完全になることが考えられる．このため，モデルの各部の機能を明確にした上で，それらを一定の構造で組み合わせるための，全体としての枠組みが必要になる．

この枠組みに従って，予め重複や欠損がないように分類されたモデルを組み合わせることで，モデルに一貫性を持たせる．又，各モデルの役割を明確にして独立性を持たせることで，再利用性を上げて拡張性を持たせることができる．

従って，設計生産活動の表現と実装を行うためには，組み合わされる各モデルの機能を明確にして，それらを組み合わせる全体的な構造を確立し，具体的な活動モデルの定義を行う必要があると考えられる．

設計から生産までの活動の管理を目的とした具体的な活動モデルの表現

さらに具体的にモデルの表現を行い，本論文の手法が有効であることを示す必要がある．モデル化を行う対象としては，本論文の対象領域の中でそれぞれ特徴を持った複数の領域を扱い，それらの活動の関連を表現する必要がある．

具体的に対象と考えられるモデルとしては，設計者がグループを構成して，相互に検討しながら設計作業を進める状況，これに対比して具体的な生産現場における活動も表現することで，本論文で提案する方法論での表現能力の有効性を示す必要がある．

すなわち，表現されるべきモデルとしては次のモデル表現を行う必要がある．

- 設計活動におけるグループ活動モデル
- 生産システムにおける製造設備の活動表現モデル
- 上記の２つのモデルの強調

2.4 形式仕様記述言語による設計生産活動表現を活用する計算機支援システムの提案

形式仕様記述言語による表現を扱う一般構造の提案

2.1から 2.3までは，設計生産の活動を表現そのものを実現するために解決すべき問題点の分析が行われた．しかし本論文の本来の目的である，設計生産分野の活動の評価や修正を行うためには，単に設計生産活動を表現するだけでなく，表現された設計生産活動を何等かの形で実装し，その結果を活用しなくてはならない．

現実的には，設計生産の活動を形式仕様記述言語を用いて表現した後で，実装する方法論に関しては，まとまった形の報告はなされていない．

従って本論文では，まず形式仕様記述言語で表現された仕様を活用する現時点での報告に関する調査を行い，どのような実装が行われているかを把握する必要がある．その後，設計生産分野で実装が可能であるか，又特有の実装方法があるかどうかを探す必要がある．

形式仕様記述言語による表現の実装方法に関する調査対象として考えられる形式仕様記述言語の種類としては，2.2で挙げた表現技法の中で，次の表現技法が考えられる，

- 通信プロトコル仕様記述法
  - SDL [若原 90] [CCI88]
  - Estele [岡田 90]
  - LOTOS [高橋 90a] [ISO89]
- 電子製品仕様記述
  - VHDL [Jame89]
- 製品モデル記述
  - EXPRESS [木村 95] [Fow95]

形式仕様記述言語による仕様表現は，その表現を人間が直接読んで理解できることがもちろん必要であるが，その記述を計算機プログラムで読み込んで直接処理を行うことができることが特徴の一つとなっており，本論文でもその実装の成果が期待される．しかし，設計生産の活動の形式仕様記述言語による表現とその実装は，いまだ一般的な用法が確立されていないのが現状である．

これらの形式仕様記述言語による表現を扱う一般的な形態を導出し，必要な機能を列挙することができれば，引き続いて行う計算機支援システムの開発においても，有効な効果を得られると考えられる．

形式仕様記述言語による表現を扱う一般構造の提案

さらに提案した基本モデルに基づくプロセスの実現など開発に際しての問題点を確認し，試作システムの開発を通じて，それらの問題点の解決を図ることが課題として上げられる．

すなわち，本論文で提案する基本モデルと形式仕様記述言語は，第3章で行ったプロセス管理機能が実現されている前提として，幾つかの機能を設けている．従って．本論文で提案した基本モデルが具体的に，計算機システム上で実現できることを検証するためには，プロセス管理機能が実際に動作可能であることを示す必要がある．

2.5 結 論

本章では，第1章であげた目的に対する問題点を個別の項目に分類整理して分析を行った．その結果，本論文が解決すべき課題として，次の問題点を得ることができた．

- 設計生産における活動に関与する，設計者や設計グループ，CAD/CAM システムやエンジニアリングデータベースなどのアプリケーションプログラム，複数の工作機械やロボットなどの製造設備，製造設備や貯蔵設備などで操作の対象となる製品，素材及び工具などの部品類，入出力されるデータ，データの構造に関するデータ (メタデータ) などを表現対象として取り扱う必要性．

- 上で挙げた様々な対象の間で，同時に進行する複数の活動を表現するための基本モデルの必要性.
- 設計生産に関係する表現対象と，その間で実行される活動を表現するための基本モデルを確立したのちに，その概念に基づく原子的な意味を規定し，それらを予め期待された構文に従って組み合わせることで，表現の対象のモデルを規定する形式仕様記述言語による表現技法を確立する必要がある.
- 上で挙げた基本モデルと形式仕様記述言語を用いた上で，対象とされる各種の活動を表現するモデルを構築するための，基本的な構成要素を組み合わせて，複数の活動を関与させることができるモデル構造を確立する必要性.
- 上で挙げた基本モデル，形式仕様記述言語，モデル構造に従った設計生産活動に関するモデルを計算機システム又は他の生産システムで用いるための方法論の確立．及び本論文で提案した基本モデルに関して定式化したプロセス管理機能の実装に関する検証の必要性.

本論文では，上記の課題に対して，基本モデル，形式仕様記述言語，モデル構造，計算機支援システムに関して 1.3.2で示した構成に従って述べる.

3　設計生産活動の表現に関する基本モデル

1.2では，設計生産の活動を表現して活用を計ることの問題点について述べた．また2.1において，その問題点の中で設計生産における活動を表現するための基本モデルに関する問題点の分析を行った．本章ではそれらの問題点を考慮した設計生産活動に関するモデルの提案を行う．このモデルを設計生産コミュニケーションモデル (Design and Manufacturing Communication Model : 以下 DMCM と呼ぶ) と名付ける．

本章の構成は，まず 3.1において既存の研究における分析を行う．その分析に基づいて，解決すべき具体的な問題点を明らかにする．その結果から設計生産の活動表現としては，イベントドリブン方式の動的モデルが有効であることを示す．次に 3.1.3において本論文で提案する基本モデルに対して重要な基盤となる，オブジェクト指向とプロセス代数に関して述べ，3.2において特定の活動における情報の流れをプロセスとして扱う設計生産コミュニケーションモデル を提案する．さらに，このモデルにおける性質など数学的な表現を行い，3.2において提案した設計生産コミュニケーションモデル と従来からの研究を比較して設計生産コミュニケーションモデルの特徴を明らかにする．

3.1　設計生産活動を表現する基本モデルに関する分析

本節では，2.1においてその重要性を認識できた設計生産活動における各種活動とその活動に関与する対象とを表現する基本モデルを確立するために，3.1.1において [BA92] の分類に基づいた既存の研究に関する分析を行い，分析結果から本論文で対処すべき課題について 3.1.2でまとめる．

本論文では形式仕様記述言語の開発を行う方針なので，従来からの定量的な生産過程の分析手法は，3.1では，対象外として扱わないこととする．

本論文の目的は設計生産活動の形式仕様記述言語による表現と実装であるが，設計生産におけるシミュレーションシステムに関して [BA92] の報告がある．この報告中の分類と [菊地 95a] で行った形式仕様記述言語に基づく設計生産の活動表現に関する調査に従って，次の項目に関する調査を行う．

- コンカレントエンジニアリングにおけるモデル
  - Clausing による活動全体に関するマクロモデル [Cla93]
  - Cleetus による設計グループの活動モデル [Cle93]
- ペトリネット
- ルールベースシステム
  - 設計対象制約言語 FDL [今村 92][今村 94]
  - 抽象実態記述言語 AbStruct[伊藤 92]
- SADT (Structured Analysis and Design Technique) 手法
  - IDEF (Integrated Computer-Aided Manufacturing (ICAM) DEFinition) [MMPd92][NISa][NISb],
  - IEM (Integrated Enterprise Modelling) [MSJ]
- 並列計算機モデル (並列論理プログラミング)

  - CIMOSA (Open System Architecture for CIM) で用いられる生産設備の仕様記述言語 MSS (Formal Specification Language for realistic integrated manufacturing systems) [Ver94] [Ver93] [Esp93]
  - EXPRESS-P (EXPRESS のプロセス表現用拡張版) [FM94] [MFF92]

- 離散的イベントシミュレーション言語
  - COSL (Concurrent Operation Specification Language) (低レベルの製造制御のための記述言語) [PK89]

[BA92] は，オブジェクト指向自体については，それ程詳しく述べていない．また本論文では他の研究分野として通信プロトコル記述に関する研究も重要視しているので，次の研究に関しては 3.1.3 で別にまとめることとする．

- オブジェクト指向
- 通信プロトコル記述 (LOTOS, Estelle, SDL)

次節 3.1.1では，各研究における基本モデルの問題点について述べる．

3.1.1 設計生産活動を形式仕様記述言語を用いて表現する既存の研究

コンカレントエンジニアリングのモデル

設計生産の活動表現に関する研究は，

- Clausing による活動全体に関するマクロモデル [Cla93]
- Cleetus による設計グループの活動モデル [Cle93]

などのように，個別の活動ではなく全体のレベルに限定したモデル化，あるいは，設計グループのネゴシエーションに関する観点で活動を抽象化したモデルなど，特定の活動に限定したモデルに関する研究から始まった．このような，特定モデルの開発は，対象としている領域の中では巧くモデルを表現することは可能であっても，その対象領域を越えて本論文で対象としたような生産活動までを含むような活動全体に関して議論及び評価を行うことは逆に非常に難しい．従って，特定の活動モデルには依存しない形で設計生産の過程を表現する必要がある．また，複数の活動が協調するような状況に対する考慮はなされていないので，本論文では，この点についての対応を考慮する必要がある．

ペトリネット

受動的システムのための方法論の中で最も一般的なモデルは，ペトリネット (Petri net) である [椎塚 92] [W.88] [奥川 95] ．ペトリネットは，システムのモデル化と解析を行うための図形的かつ解析的な方法論である．ペトリネットを設計生産活動の表現に用いた例として，[BA91] から複数ロボットによる並行活動をモデル化した研究が報告されている．

しかしペトリネットは，受動的モデリングツールとしては，記述能力が制限されており，明確な時間に関する情報が表現できない点が指摘されている [BA92]．それらの限定事項を克服するために，ペトリネットモデルに対する多くの拡張が提案された．幾つかは，時間情報を陽に持つ形の拡張，他

はトークンのバリエーションに関するものである．拡張モデルは，通信プロトコルの確認，シミュレーション，ソフトウェア開発，生産モデリング，その他に用いられている [BA92].

ここでは，[奥川 95] から，ペトリネットの数学的な定義を一部引用する．有効グラフのノード集合 Vを V_1と V_2に分割する． V_1をプレースの集合 P，V_2をトランジションの集合 Tとする．アーク (弧) は，プレースからトランジションに向かうアーク $\{P \times T\}$ とトランジションからプレースに向かうアーク $\{T \times P\}$ のどちらかしか存在しない．

ペトリネットの構造 Nは，状態変数を表わすプレースの有限集合 P，状態遷移を示すトランジションの有限集合 T，情報や制御の流れを示すアークの有限集合 A

$$P = \{P_1, P_2, \ldots, P_n\} = \{P_i | 1 \leq i \leq |P| = n\} \quad (1)$$

$$T = \{T_1, T_2, \ldots, T_n\} = \{T_j | 1 \leq j \leq |T| = n\} \quad (2)$$

$$A \subseteq \{P \times T\} \cup \{T \times P\} \qquad (P \cap T = 0, P \cup T = 0) \quad (3)$$

を使った 3 項組として集合的に式 (4) と表せる．

$$N(P, T, A) \quad (4)$$

最も基本的なペトリネットは，上記のプレースとトランジションにトークンが存在する形でシステムの状態を表わす．従ってシステムの状態そのものは，(プレース + トランジション) × トークンの組み合わせだけ存在する有限状態機械として考えることができる．しかし，遷移則に関しては外的要因によってトランジションでの発火が制限されるモデルがあるので，常に一定の遷移則が用いられるわけではない．

ペトリネットでは，上記の基本構造に対する幾つかの拡張は見られるものの，その分析は，アークの接続関係をグラフとして表現するマトリックスを用いた分析で行われる．従って，この分析では接続関係 (アークの構造) は，変化しないことが大前提となっている．

ルールベースシステム

次に，従来からある設計問題の解決のために，ルールベースシステムに基づく形式仕様記述言語を用いてルールを記述する報告が幾つかなされている．

- 岡田 [岡田 92] による，設計問題決定の過程を 2 進木で表現することでの問題解決過程の表現を目的とする設計過程記述言語 DPL (Design Process Language)
- 今村 [今村 92][今村 94] による，設計対象に関する制約事項を表現する言語 FDL
- 伊藤 [伊藤 92] による，幾何的な位置関係及び拘束関係などを記述するための抽象実体記述言語 AbStruct

[今村 92] や [伊藤 92] の研究では，対象となる製品あるいは問題に関する制約条件を表現するために形式仕様記述言語を用いている．[岡田 92] の研究では，形式的な表現技術は用いていないので，表現された活動モデルの実装には疑問が残る．

この制約事項は，記述論理に基づく IF-THEN 形式の制約表現に基づいて行われ，これを問題解決器への入力とする．通常ルールベースモデリングシステムと呼ばれる．

問題解決器は，LISP や PROLOG などの記述論理言語のインタープリタを用いて開発されており，そのインタープリタのパターンマッチングのメカニズムを利用して，入力された制約事項に沿って許容される解を探索して問題解決を行う．

ルールベースモデリングシステムでは，システムは物理的ないしは演算的な規則として表現される．このアプローチは，システムの詳細がはっきりしていない状態で，システムの機能的な視点を強調するためには有効であることが判明している．

これらの言語による表現は，基本的には対象における静的なかつ強制的な制約の表現が行われ，対象の動的な変化に関する表現は行われない．設計や生産における具体的な活動表現の視点で見ると，これらの研究における問題解決の過程は，対象領域や並行した活動表現には不十分と考えられる．その理由は，設計解の解決モデルのための制約表現としての意味はあるが，システム内部では問題解決器の中で記述されている設計解の探索アルゴリズムでしか動いていない点にある．そのため解の探索を行う以外の目的での実装が難しいと考えられる．

SADT (Structured Analysis and Design Technique) 手法

SADT 手法に関しては， 1.2.2節において述べたのでここでは割愛する．

並列計算機モデル (並列論理プログラミング)(CIMOSA)

SADT 手法に対して，階層的でない制御システムが提案された．これらのアプローチでは，分散的かつ並列実行のモデリング能力が要求され，これらの提案されたモデリングの方法論は，並列計算機モデル (並列論理プログラミング) を用いている [GR89][ナイ 89]．

CIMOSA は，ESPRIT プロジェクトのシリーズの中の EP 688，5288，7110 として比較的最近に始められたプロジェクトで，プロセスの表現に関しては新しい視点に立って行っている．その目的として，次の項目が上げられる [Ver93]．

- 実行可能な企業モデルを導出する企業活動モデリングの枠組みを提供する．
- ビジネスの統合化及び企業活動のオペレーションをモニターして制御するための，インプリメンテーションモデルの実行のサポートのために用いられる，企業活動モデリングの枠組みを提供する．
- CIM システムのライフサイクルを記述する方法論を提供する

CIMOSA は，プロセスの集合としてモデル化する形式仕様記述言語 MSS を用いて CIM システムの仕様を定義するために策定された [Esp93]．MSS は，モデルの交換のための統合化された製造のシステム仕様，モデルの発展，及びモデル交換のための中立フォーマットのための形式的な記述方法になることを目的とした形式仕様記述言語である．言語は CIMOSA の提案するプロセスモデルに基づいており，機能的情報とリソースアスペクトを記述対象とする．

モデルは時間だけでなく情報，材料，リソースオブジェクト仕様を考慮に入れたイベントドリブン方式である．その目的は，本質的な企業のアスペクトを一つの統合化されたモデルでカバーすることである．これらのアスペクトには，次の４点がある．

Functional Aspect 入力に対する出力としての関数的機能の視点．

Behavioural Aspect イベントやビジネスルールと呼ばれ，現在のシステムの状態に従って，どのような出来事が起こるかの視点．

Resource Aspect 要求される物理的な意味及びそれらの管理の視点.

Information Aspect 企業活動を実施する際に用いられ交信される知識

Organizational Aspect システムの構成部分上で反応し，担当者を決定する，決定のレベル及び決定の中心の視点.

システムの記述は，ドメインと呼ばれるモデュールに分けて行われる．ドメインは最低一つのドメインプロセスを持ち，そのドメインが受け取れるイベントと起動されるアクションとによって定義される.

CIMOSA でのプロセスの概念は非常に現実的なものであるが，次のような点が不十分と考えられる.

- プロセスの概念は手続き的な関数に近いと言えるが，プロセスそのものの数学的な定義が明確に行われていない.
- プロセスがドメインに属する形でしかモデル化されていない．従って，イベントが発生してドメインからプロセスが起動される際のプロセスの一意性の保証，その中での並行性や，複数のプロセス同士の順序性に関する情報は表現できるようになっている.

離散的イベントシミュレーション言語 (COSL)

コンピュータのプログラム言語は，システムモデリングとシミュレーションのために広く用いられている．伝統的な離散的イベントシミュレーション言語に加えてオブジェクト指向言語やルールベースシミュレーション，分散シミュレーションアプローチが行われた．分散シミュレーション又は制御用言語は，非同期プロセスのネットワークやタイムスタンプのメッセージを用いる通信に基づいている．これらの例は，COSL[BB93] や DISS などがある．この方向での実装は，ドメインを特定してしまい，かつ中央集権的なタイムシェアリングコンピュータの環境に近いと言える．それらのツールの殆どは，現実の並列モデリングはサポートしておらず，コンフリクトを解消するためやリソース管理に特殊なメカニズムを持っている.

COSL (Concurrent Operation Specification Language) [PK89] は，製造時における低レベルのシステム制御の表現を目的とした記述言語である．この言語では，全体の活動はオペレーションの集合として定義される．オペレーションは，条件とアクションの集合として定義される．各々のオペレーションは，識別のための一意な整数番号とオペレーション内部で実行されているコントロールステップを記述するオペレーション名とが与えられる．あるオペレーションのための条件が真となった場合に，そのオペレーションに関する反応が実行される．条件と反応の組み合わせによって，実行中のオペレーションが完了する前に，連続してオペレーションが起動されることが可能である.

COSL では，低レベルの制御の表現を目的としているため，次のような問題点を上げることができる.

- 同時に並行して実行されるプロセス同士の関係に関する概念が定義されていない.
- 入出力されるメッセージが単純なシグナルとして表現されているため，意味的に高度な情報の入出力を行うことが難しい.
- 反応及び出力が，低レベルの入出力として予め規定されている，従って意味的には低レベルのものしか表現できない.

通信プロトコル記述 (LOTOS, SDL, EXPRESS-P)

設計生産分野以外のプロセスのとらえ方として，ソフトウェア工学におけるオブジェクト指向，及び通信又は計算機分野におけるプロセス代数がある．オブジェクト指向的な見方では，プロセスを「周囲のプロセスや周りの環境と通信を行う抽象的なエージェント」としてとらえることができる [二木 92]．この概念での周りの環境とは，プロセス以外のすべての入出力対象を指し，通信の対象という意味でプロセスと同等に扱われる．

情報通信分野では，OSI の階層モデルを記述するために LOTOS 及び SDL が用いられている [二木 90]．LOTOS は，プロセス間の情報の入出力を記述する以外に，ACT ONE に基づいて入出力されるデータの型及びその型に対して適用される演算を記述する能力を備えている．SDL 及び LOTOS などの通信プロトコル記述仕様言語は，プロセス代数の考え方に基づいて定義されている．プロセス代数は，プロセスの数学的な定義を目的として構築された代数系であり，オブジェクト指向的な考えに加えて，観測可能性及び双模倣性，時間的関係を重要視する．プロセス代数では，プロセスを「対象の行動パターンを記述する，周りの環境から観測されるイベント (情報の入出力の出現) の有限集合」として定義する．非常に単純な基本モデルに基づいており，一般性及び拡張性に富んでいる．

EXPRESS-P [FM94] [MFF92] は，STEP で製品モデルのデータ構造を定義するために用いられている EXPRESS をプロセス表現用に拡張したものである．言語仕様としては，SDL (Specification and Description Language) [CCI88] 的な拡張が行われ，各エンティティの入出力定義とそのメッセージに対するアクションとを定義することができる．また図式表記をおこなうために Estelle [ISO91], GRAPES-86 [DH92] と同じような拡張を行っている．

EXPRESS-P での情報の入出力関係は，SDL 的な通信回線の接続のように静的な接続関係で表現されている．即ち，あるエンティティから入出力が行われる相手のエンティティの種類は，そのエンティティ内で属性表現の一種として決められており，実際に接続されるインスタンスは物理ファイル上で定義される．このため，次のような問題点が上げられる．

- 情報を入出力する相手の種類が固定されているので，多様な情報の入出力まで表現できるか疑問が残る．
- 属性としての参照と，情報の入出力先としての参照とが混同されるか疑問が残る．
- 入出力先が 1 種類であっても，その中で設計グループの構成が動的に変化するような活動を表現できるか疑問が残る．
- STEP のアーキテクチャ内部の問題ではあるが，定義されたエンティティ(及びプロセス) は，物理ファイルにインスタンス化することで表現される．実際には，どのようなメカニズムでプロセスの実行が行われるかが不明である．

ここまで，形式仕様記述言語を用いて設計生産のプロセスを表現する研究に対する考察を行ってきた．次節 3.1.2では，これらの考察をまとめて本論文における課題を整理する．

3.1.2 本論文における設計生産活動に関する基本モデルへの課題

3.1.1で既存の各研究に対して行った考察から，設計生産活動の表現を行うためにどのような課題を解決する必要があるかについて整理する．問題となった点は，まず全体的な概念，次に多様な対象

をどのように表現するか，プロセスを規定する基本モデル，入出力される情報，などであった．前節でそれぞれの研究に対して上げた課題全体を，これらの項目毎にさらに細目に関してまとめる．

1. 全体的な構成について

   - 特定の活動モデルを前提として，そのモデルに依存する形での設計生産の過程表現ではなく，ある程度複数のタイプのモデルでも，表現して組み合わせることができることを目標とする．例えば，制約条件の述語論理による表現と探索アルゴリズムの組み合わせによる表現は，そのモデルへの依存性が高く他のモデルへの活用が難しい．

2. プロセスに関する概念

   - SADT 手法的な箱の入れ子構造のドメインにプロセスが属する方式では，プロセスの前後関係や並行関係を表現できないので，より動的にプロセスが活動する表現を行う．
   - 設計生産における活動を独立した存在として陽に表現し，時間的な特性，他の活動との並行活動や時間的な前後関係，親子関係，複数の活動の協調などを記述する．
   - プロセスの記述は，手続き的な関数に近い表現でもよいが，数学的な明晰性を持ち，形式仕様記述言語の意味論的な仕様を規定しやすい基本モデル である必要がある．

3. 対象の表現能力

   - 設計生産活動に関係する設計活動者と設計グループ，CAD/CAM システムなどのアプリケーションプログラム，NC 工作機械，ロボットなどの生産設備などの幅広い対象を表現する記述能力．
   - 各活動において扱われる情報，例えば STEP の製品モデルなどを上記の対象と同様に表現する能力．

4. プロセス及び対象間での入出力

   - 情報を入出力する相手の種類を固定しないで，実行中に自由に変更できる方式にする．入出力先の種類が固定されている場合でも，構成が動的に変化するような活動を表現できるようにする．
   - 入出力される情報や反応などを，シグナルの出力やレジスタ／カウンタ値の変更などの低レベルで単純なシグナル的な表現ではなく，より高度な意味を持つ情報の入出力として行う．

5. プロセスの実行メカニズム

   - 表現されたプロセスが，実際にどのようなメカニズムで実行が行われるかを考慮した設計にする必要がある．

設計生産の活動を表現するためには，上で挙げた問題点を克服できるモデルが必要になる．ここで挙げた課題を解決を目指した 3.2において，プロセスに関する基本モデルの提案を行う．

3.1.3 オブジェクト指向，プロセス代数及び通信プロトコル記述言語の特徴

本論文では，プロセス代数とオブジェクト指向的な概念に基づいたプロセスの定義を用いて，3.1.2で挙げた問題点の解決を計る．本節では，それらの特徴を予め述べておく．プロセス代数には，数種類の系統があるが，本論文ではCCSに基づいた概念の表現を用いるので，特にCCSに関して重点的に報告を行う．

オブジェクト指向 (Object Oriented Methodology)

オブジェクトとは，ある状態を共有する操作の集まりである．操作は，オブジェクトが応答できるメッセージを決定し，共有状態は外部世界から隠蔽され，そのオブジェクトの操作によってのみアクセスすることができる．オブジェクトの内部状態を表わす変数は，インスタンス変数 (Instance Variable) と呼ばれ，操作はメソッド (Method) と呼ばれる．メソッドの集まりがオブジェクトのインタフェース (Interface) と振る舞い (Behaviour) を決定する [Bud92][Weg92]．通常メッセージの受け渡しには，カプセル化が用いられる．つまりメッセージを送る側では，そのメッセージによる要求が実際にどのように実行されるかは問題とされない．

オブジェクト指向では，オブジェクトのインスタンスを生成するテンプレートとして，クラスの概念を用いる．また，新たなクラス定義において他のクラスでの定義内容を再利用するために，継承 (Inheritance) が行われる．オブジェクト指向では，この継承を使って似たような特性のクラスをまとめて階層構造にしてモデリングを行う．また，実際にはインスタンス化されることはないが，サブクラスを導くためだけに用いられるクラスを抽象クラスと呼ぶ．

サブクラスでは，継承したメッセージを，そのクラスの内部変数に適応した形で再定義を行う．こうすることで，実際には異なるクラスに対して同じメッセージを送って実行させることが可能になる．これを多様性 (Polymorphizm) と呼ぶ．

オブジェクト指向的な見方では，プロセスを「周囲のプロセスや周りの環境と通信を行う抽象的なエージェント」としてとらえることができる．この基本モデルでの周りの環境とは，プロセス以外のすべての入出力対象を指し，通信の対象という意味でプロセスと同等に扱われる．

オブジェクト指向言語は，それらの構造的かつ機能的なシステムアスペクトからデータモデルの構築に有効である．生産システムモデリングでは，モデリングで並列実行を要求しない典型的には階層的な制御と操作の元で用いられる [BA92]．

プロセス代数 (Process Algebra)

プロセス代数 (Process Algebra) とは，現実世界でのプロセス (あるいはプロセスと呼ばれるもの) を，構文的には形式体系 (Formal System)，意味論的には代数としてモデル化した数学的対象である．従ってプロセス代数におけるプロセスとは，その形式体系を構成する要素としてのみ厳密に定義される [二木 92]．

これらの代数系は，

- システムの状態をモデル化した動作式または挙動式 (Behaviour expression)
- 動作式の間の遷移関係を定義する遷移規則 (transition rule)
- 等価性によって定まるモデル

などの定義方法の違いによって区別することができる．これらの違いに応じて，

- CCS (Calucuas of Communicating Systems)[Mil89]
- CSP (Communicating Sequential Processes) [Hoa85] [Hoa92]
- ACP (Algebra of communication Process) [BK85]

が提案されている．各理論の詳細について完全に述べることは不可能であるが，以下では，本論文で提案する基本モデルを定式化するために必要と思われる部分について述べる．

C C S

ここでは，[Mil89] による原著と [富樫 93a] による解説とを元にして CCS の解説を行う．原著全体を引用することはできないので，後に行う定式化が理解できる範囲で，ごく基本的な部分について解説を行う．

CCS では，その回りの環境 (他のプロセスも一つの環境になる) と通信を行うエージェントとしてプロセスを捉える．プロセスは，データを入出力するためのポートと，それぞれのポート固有のラベルとを持つ．以下ではポートのラベルと，そのラベル上でのアクションは同じ意味で扱われる．すなわち，外部から観測可能なプロセス上の出来事がイベントであり，そのイベントは同じ名前のポートで発生するプロセスのアクションとして捉えられる．プロセスの内部動作は表現されずに，その挙動はプロセスのポートで発生するアクションの順番で表現される．

引き数を伴った一つの入力 in を受け取り，同じ値の出力 out を行う単純なプロセスのモデル C は，式 (5) で表現される．

$$C \stackrel{def}{=} in(x).\overline{out}(x).C \tag{5}$$

この時 x は任意の値を持つ引き数である．$\overline{out}$の上線は，別のプロセスの別のポートに対応している相補的なアクションであることを示しており，その意味は後で詳述する．図 7にプロセスとポートの基本モデルを示す．

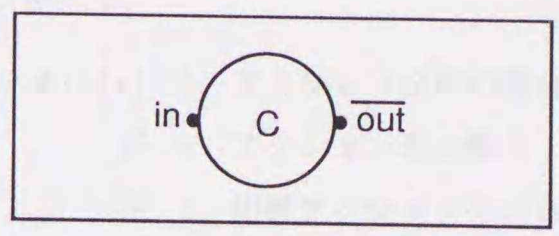


図 7: CCS でのプロセスの入出力ポート

この式表現は，Cの挙動を再帰的に表現している．すなわちプロセス Cは，入力ポート in から x を受け取り，出力ポート out から x を出力して，初めの状態に戻り同じ挙動を無限に続ける．この Cの挙動は，入出力ポートにおける情報の交換のみを通じて表現されていることになる．CCS では，ポートにおけるアクションをアクション名と相補アクション名に分けている．アクション名全体の集合を Nとし，含まれる個別のアクション名をα，β などのギリシャ文字を用いて表す．

CCS では，各アクションαに対して，同期通信の相手となる相補アクション名 (co-name) $\overline{\alpha}$を仮定する．相補アクションの集合は，式 (6) の形で定義される．

$$\overline{N} = \{\overline{\alpha} | \alpha \in N\} \tag{6}$$

相補アクション名$\overline{\alpha}$ の相補的なアクションをαとする．すなわち，$\overline{(\overline{\alpha})} = \alpha$となる．$A = N \cup \overline{N}$ の要素をラベルと呼びa，b，cなどで表わす．L，KはAの部分集合を表わすのに用い，その相補的な部分集合は式 (7) で表される．

$$\overline{L} = \{\overline{\alpha} | \alpha \in L\} \tag{7}$$

ラベルはシステムによって実行されるアクションであり，外部からは観測できるとする．図 8に相補アクションとして定義された，並行動作する２つのプロセスとポートの関係を示す．この図では，プロセスAとBは並行に動作しており，Aのアクション$\overline{out}$とBのアクションoutとが相補的な関係になっていることを示しており，２つのプロセスそれぞれは並行に動作しているが，アクションoutの発生に関してだけ２つのプロセスに同期したイベントが発生することを示している．

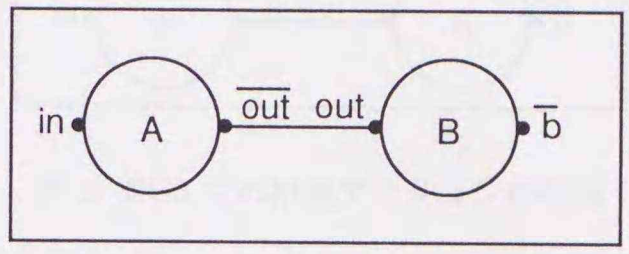


図 8: CCS での２つのプロセスの相補アクション

CCS には，A の他に A のアクションとは異なる特別なアクションτ (内部アクション (internal action) 又はサイレントアクション (silent action) と呼ばれる) がある．τ は，システムの内部動作に対応し，システムの外部からは観測できず，制御する事もできない．A と τ を合わせてアクションと呼び，

$$Act = A \cup \{\tau\} \tag{8}$$

とする．アクションを表わすのに，従来と同じようにa，b，cなどの記号を用いる．CCS のアクションについてまとめると，次のようになる．

$$\begin{aligned} Act = A \cup \{\tau\} &\ : \text{アクション全体の集合} \\ N &\ : \text{アクション名の集合} \\ \overline{N} &\ : \text{相補アクション名の集合} \\ A = N \cup \overline{N} &\ : \text{ラベルの集合} \\ \tau &\ : \text{内部アクション} \end{aligned}$$

次に A から A への関数 $F : A \to A$ で

$$f(\overline{a}) = \overline{f(a)} \tag{9}$$

を満たす関数をリラベリング関数と呼ぶ．このリラベリング関数は，特定のアクションのラベルを付け替えて他のアクションのラベルに置き換えることを行う．上の式は特定のアクションに対してリラベリングを行っても相補的な関係は維持されることを示している．さらに便宜的に $f(\tau) = \tau$とし，リラベリング関数 fをアクションの集合全体まで拡張する．

図 9に相補アクションとして定義された，並行動作する２つのプロセスとポート制限 (restriction) を行った状態を示す．この図では，プロセス A と Bの Aのアクション$\overline{out}$と Bのアクション out とが

相補的な関係になっていることを示しており，そのアクション *out* の発生を制限した状況を示している．制限とは，イベントの発生を外部に対して隠蔽してしまい観測不可能にしてしまうもので，結果的にその２つのプロセスにおけるイベントの発生を外部から観測することが不可能になってしまう．

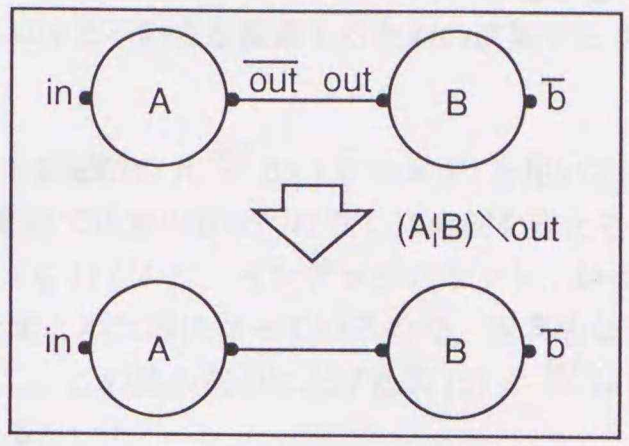


図 9: CCS での相補アクションの制限

次にプロセス動作式の定義を行う．初めにプロセス変数 x，y，... を仮定する．Iや Jは，インデクシングセットであり，Iによって順序付けられる数式のファミリに対して $\{E_i : i \in I\}$ として用いる．また，順序付けられたファミリ $\{E_i : i \in I\}$ の Iが理解されている時には，$\{E_i : i \in I\}$ を$\tilde{E}$で置き換える．例えば，$\sum\{E_i : i \in I\}$ は $\sum \tilde{E}$に置き換えられる．また，数式中の変数を置き換える場合，$\{X_i : i \in I\}$ 又は$\tilde{X}$ を変数の順序付けられたファミリとする．この時，全ての $i \in I$による X_i の全ての出現の E_iによる同時並行した置き換えの演算は，$\{E_i/X_i : i \in I\}$ 又は $\{\tilde{E}/\tilde{X}\}$ によって表現される．

プロセス動作式は，次のように定義される．

1. プロセス変数 x は，プロセス動作式である．このプロセス変数には，アクションなどで構成されるプロセス動作式を代入することができる．

2. 無動作プロセス 0 (*Nil*と書くこともある) は，プロセス動作式である．

3. Eがプロセス動作式であるとき，アクション $a \in Act$ によるアクションプレフィックス (action prefix) $a\,.\,E$ は，プロセス動作式である．この時のアクションプレフィックス (ドット表記) は，イベント a が発生し，そのプロセス動作式が Eに遷移することを示す．

4. $+$は，２つの引数であるプロセス動作式の内のどれか一つが発生することを示す．E_iをプロセス動作式とすると，これらを$+$によって結合したインデクシングセット I上での総和 (summation) $\sum_{i \in I} E_i$ はプロセス動作式である．

5. E_1，E_2をプロセス動作式とすると，それらの $|$ による合成 (composition) $E_1|E_2$ はプロセス動作式である．この式は，２つのプロセス動作式が並行して動作することを示す．

6. E をプロセス動作式とすると，ラベルの集合 $L(L \subset A)$ による制限 (restriction) $E \setminus L$ はプロセス動作式である．この時 L に属するアクションは，隠蔽されてしまい外部から観測不可能になる．

7. E をプロセス動作式とすると，リラベリング関数 (relabelling) $E[l_1/l_1', \ldots, l_m/l_m']$ は，プロセス動作式である．この関数は，Eのアクション l_1 を l_1' に，...， l_m を l_m'に置換する．

8. x_1，...，x_n をプロセス変数 E_1，...，E_n をプロセス動作式とすると，再帰 (recursion) $\mu x_j.\{x_i = E_i | 1 \leq i \leq n\}$ は，プロセス動作式である．ここで jは $1 \leq j \leq n$ の任意のインデックスである．μは再帰呼び出しにおける不動点を探索するための演算であり，再帰的ではない表現を行っていることになる．

[Mil89] の報告では，再帰は別の表記法の $A \stackrel{def}{=} \mathbf{fix}\,(X = a.X)$ を用いて表現している．$\mathbf{fix}\,(X = a.X)$ は，**fix** 式と呼ばれ Xは境界変数で任意の意味づけなしに名前を変えることができる．この表記で再帰の定義は，$\mathbf{fix}_j(\{X_i = E_i : i \in I)$ (I は，インデックスセット，かつ $j \in I$) となる．ただしここでの Eは，複数のイベントをまとめた形になっているので，現実的な再帰では相互に再帰呼び出しが行われることになる．従って，この節の初めに上げた式 (5) $C \stackrel{def}{=} in(x).\overline{out}(x).C$ は，その意味では不完全な表現であると言える．

次に遷移規則について述べる．CCS では，プロセスは同期木又はプロセスグラフと呼ばれる遷移システムであり，外部観測による等価性に基づいた判断が行われている．プロセス全体は一つの遷移システムであり，その意味は遷移関係によって与えられるものである．ここでの遷移システムは，次の式 (10) に示す一般的なラベル付けされた遷移システムの表記を用いる．

$$(S, T, \{\stackrel{t}{\rightarrow} : t \in T\}) \tag{10}$$

は，状態の集合 S，遷移ラベルの集合 T，各々の $t \in T$ごとの遷移関係$\stackrel{t}{\rightarrow} \subseteq S \times S$ で定義される．CCS の表記では，状態 Sはプロセス動作式が，遷移ラベル Tにはアクションが対応する．プロセス動作式上での遷移関係は，$\stackrel{\alpha}{\rightarrow}$ となる．例えば並行動作による推移規則を導出する場合は，$A \stackrel{\alpha}{\rightarrow} A'$ から $A|B \stackrel{\alpha}{\rightarrow} A'|B$ が推測される場合は，一般規則は式 (11) のようになり

$$\textbf{From} \quad E \stackrel{\alpha}{\rightarrow} E' \quad \textbf{infer} \quad E|F \stackrel{\alpha}{\rightarrow} E'|F \tag{11}$$

この表記を以下では式 (12) で示す．

$$\frac{E \stackrel{\alpha}{\rightarrow} E'}{E|F \stackrel{\alpha}{\rightarrow} E'|F} \tag{12}$$

CCS の場合，プロセスの遷移関係は次の遷移規則によって与えられる．

$$\frac{}{a.E \stackrel{a}{\rightarrow} E} \quad Act \quad (Action) \tag{13}$$

$$\frac{E_j \stackrel{a}{\rightarrow} E_j'}{\sum_{i \in I} E_i \stackrel{a}{\rightarrow} E_j'} \quad Sum_j\ (j \in I) \quad (Summation) \tag{14}$$

$$\frac{E \stackrel{a}{\rightarrow} E'}{E|F \stackrel{a}{\rightarrow} E'|F} \quad Comp1 \quad (Composition) \tag{15}$$

$$\frac{F \stackrel{a}{\rightarrow} F'}{E|F \stackrel{a}{\rightarrow} E|F'} \quad Comp2 \quad (Composition) \tag{16}$$

$$\frac{E \stackrel{a}{\rightarrow} E' \qquad F \stackrel{\overline{a}}{\rightarrow} F'}{E|F \stackrel{\tau}{\rightarrow} E'|F'} \quad Comp3 \quad (Composition) \tag{17}$$

$$\frac{E \xrightarrow{a} E'}{E \setminus L \xrightarrow{a} E' \setminus L} \quad Res(a, \overline{a} \notin L) \quad (Restriction) \tag{18}$$

$$\frac{E \xrightarrow{a} E'}{E[f] \xrightarrow{f(a)} E'[f]} \quad Rel \quad (Relabelling) \tag{19}$$

$$\frac{E_j\{\mu\tilde{x}.\tilde{E}/\tilde{x}\} \xrightarrow{a} E'}{\mu x_j.\tilde{E} \xrightarrow{a} E'} \quad Rec \quad (Reccursion) \tag{20}$$

$$\frac{P \xrightarrow{a} P'}{A \xrightarrow{a} P'} \quad (A \stackrel{def}{=} P)\ Con \quad (Constant) \tag{21}$$

なお [Mil89] の報告では再帰における遷移規則は.

$$\frac{E\{\mathbf{fix}(X = E)/X\} \xrightarrow{a} E'}{\mathbf{fix}(X = E) \xrightarrow{a} E'} \quad Rec \quad (Recursive) \tag{22}$$

で示されている．しかし意味的には等しいものと考えられる.

$E_1|E_2$ は，E_1 と E_2 の間にポート間の接続が結ばれたプロセスを表わす．しかし，この接続関係は互いに相補的なポートラベルの間だけになる．$Comp1$ と $Comp2$ とは，E_1 と E_2 とが一方のプロセスとは関係なく一方的にアクションを実行することができることを示す．$Comp3$ では，$F \xrightarrow{a} E'$ と $F \xrightarrow{\overline{a}} F'$ の a と$\overline{a}$とが丁度互いに相補的なアクションを行うことができる場合に両者が同期して行動する．この形式で２つのアクションの同期した動作を表現することができ，この時だけこの動作は外部から観測不能になる.

$Comp1$ と $Comp2$ とにより合成演算子で非同期システムを記述することができる．しかし同期した通信は，相補関係にある１対１の通信のみであり，CSP や LOTOS のように１対多の通信を行うメカニズムは CCS には用意されていない．この点は，本論文における問題点となると考えられるが，その問題含めてＣＣＳを用いた定式化を 3.2.2において行う.

CCS における等価式

CCS には，0 と再帰を除いて５つのプロセス演算子がある．これらは，静的等価式と動的等価式に分けられる．合成，制限とリラベリング演算子が静的等価式に関係し，アクションプレフィックスと和の演算子が動的等価式に関与する．本論文では，等価式に基づいたプロセスの簡約化までは踏み込めないので，ここでは，等価式を列挙しておくにとどめる.

プロセスに対する静的等価式 (static equation) には，次のものがある.

$$p|q \sim q|p \tag{23}$$

$$p|(q|r) \sim (p|q)r \tag{24}$$

$$p|0 \sim p \tag{25}$$

$$p/L \sim p \quad (\text{ここで } \mathcal{A}(p) \cap (L \cup \overline{L} = \phi) \tag{26}$$

$$p \setminus K \setminus K \sim p \setminus (K \cup L) \tag{27}$$

$$p[f] \setminus L \sim p \setminus f^{-1}(L)[f] \tag{28}$$

$$(p|q) \setminus L \sim (p \setminus L)|(q \setminus L) \quad (\text{ここで } \mathcal{A}(p) \cap \overline{\mathcal{A}(q)} \cap (L \cup \overline{L}) = \phi) \tag{29}$$

$$p[Id] \sim p \tag{30}$$

$$p[f] \sim p[f'] \quad (\text{ここで } f \uparrow \mathcal{A}(p) = f' \uparrow \mathcal{A}(p)) \tag{31}$$

$$p[f][f'] \sim p[f \circ f'] \tag{32}$$

$$(p|q)[f] \sim p[f]|q[f] \quad (\text{ここで } f \uparrow (L \cup \overline{L}) \text{ は単写であり, } L = \mathcal{A}(p|q) \text{ とする}) \tag{33}$$

C S P

ここでは，富樫による解説 [富樫 93b] を引用する．CSP のプロセスは，内容が分からないブラックボックスであり，外部からの試験によって表現される．CSP の場合はデッドロックの可能性に着目して，失敗 (failure) の概念を用いてプロセスの意味を定義している [富樫 93b]．プロセスはブラックボックスなので，その中身は見えないが，イベントの実行可能性だけは見えるとする．

$$p \xrightarrow{t} q,\ t \in \mathcal{A} \tag{34}$$

式 (34) で t は，プロセス p のある時点までの観測可能なトレース (trace)(イベント系列) であり，その後の p の振る舞いは q と同様である．A は，イベント t の集合である．ただし，p や q の具体的な中が観測される訳ではない．遷移関係については，次の規則が仮定される．

(L1) $p \xrightarrow{s} q, q \xrightarrow{t} r$ ならば $p \xrightarrow{st} r$

(L2) $p \xrightarrow{st} r$ ならば $\exists q.(p \xrightarrow{s} q, q \xrightarrow{t} r)$

(L3) $p \overset{<>}{\rightarrow} r, q \overset{<>}{\rightarrow} p \Leftrightarrow p = q$

(L2) は，st がプロセス p のトレースならば，p がイベント系列 s を実行した後の状態を表す中間プロセスが必ず存在し，その後 t を行うとプロセス q にたどり着くとする規則である．**(L3)** は，空系列 (空トレース) に関して規定している．空系列は外部から観測不可能な特別なイベントである．空系列による遷移には２つの可能性がある．一つは，プロセスの状態が遷移しない場合で

$$p \xrightarrow{t} p \tag{35}$$

もう一つは，内部動作に当たる空系列によってプロセスの動作が進行し，状態遷移をおこす場合で，$p \neq q$ の場合の遷移関係

$$p \overset{<>}{\rightarrow} q \tag{36}$$

がこれに相当する．この場合 p は，外部から観測不可能な内部イベントによって非決定的にその状態を q に変える．この遷移は，これまでの τ による遷移同様，外部から制御できない．

最初のステップで実行可能なプロセス p のイベントの集合を

$$initials(p) = \{a \in \mathcal{A} | \exists q.(p \xrightarrow{t} q)\} \tag{37}$$

と置く．ここで $< a >$ は１個のイベント a だけからなる系列を表す．プロセスがイベントを実行できるのは，外部の環境がそのイベントを提供した場合だけであった．従って，現在外部の環境が提供可能なイベントの集合を X とすると $initials(p) \cap X$ に属するイベントだけが等実行可能となる．この集合が空ならば，環境は X 以外のイベントは提供できないし，p の方は X のイベントは実行できないから，プロセス p の進行は永久に止まってしまいデッドロックになる．

これ以外にデッドロックが起こる可能性としては，

$$refusals(p) = \{X | X \subset \mathcal{A}, \exists q.(b \overset{<>}{\rightarrow} q, initials(q) \cap X = \phi)\} \tag{38}$$

がある．$X \in refusals(p)$ の時，X は p の拒絶 (refusals) と呼ばれる．拒絶に関しては，次の性質が成り立つ．

(R1) $\phi \in refusals(p)$

(R2) $Y \in refusals(p)$ かつ $X \subset Y$ならば, $X \in refusals(p)$

(R3) $X \in refusals(p)$ かつ $T \subset (\mathcal{A} - initials(p))$ ならば $X \cup Y \in refusals(p)$

CSP でのトレースは次の定義式で与えられる.

$$traces(p) = \{t \in \mathcal{A}^* | \exists q.(p \xrightarrow{t} q)\} \tag{39}$$

(T1) $<> \in refusals(p)$

(T2) $st \in traces(p)$ ならば $s \in traces(p)$

プロセス p の失敗 (failure) とは, トレース $t \in traces(p)$ と拒絶 $X \in refusals(p)$ のペア (t, X) として定義され, プロセス p の動作がイベント系列 t を実行することによってある状態まで進行したとき Xのイベントについては反応できない状況を表す. すなわち, トレースを考慮したデッドロックの可能性である. このような失敗の集合を

$$failures(p) = \{(t, X) | \exists q.(p \xrightarrow{t} q, X \in refusals(q))\} \tag{40}$$

となる. 定義から $F = failures(p)$ について次の性質が成り立つ.

(F1) $(t, X) \in F$ならば $t \in \mathcal{A}^*, X \subset \mathcal{A}$

(F2) $(<>, \phi) \in F$

(F3) $(st, \phi) \in F$ならば $(s, \phi) \in F$

(F4) $X \subset Y, (t, Y) \in F$ならば $(t, X) \in F$

(F5) $(t, X) \in F, (t, < c >, \phi) \notin F$ならば $(t, X \cup \{c\}) \in F$

さらにプロセスに関して次の規則が仮定される.

(L4) $(F1)$〜$(F5)$ を満たす Fに対して, $failures(p) = F$なるプロセス p が存在する.

(L5) $failures(p) = failures(q)$ ならば $p = q$

A C P

ACP は, 上記の 2 つの理論とは異なり, 幾つかの演算子が定義されていて, かつ等式の集合を満たす代数としてプロセス代数を定義している. 基本システムと呼ばれる BPA (Basical Process Algebra) では, 2 項演算子 “ + ” と “ · ” 及び, いくつかの定数 $a, b, c \ldots$ 及び次の 5 つの等式からなる代数的理論である.

$$x + y = y + x \tag{41}$$

$$(x + y) + z = x + (y + z) \tag{42}$$

$$x + x = x \tag{43}$$

$$(x + y)z = xz + yz \tag{44}$$

$$(xy)z = xyz \tag{45}$$

ここで，定数 $a, b, c \ldots$ は，アトミックアクション (atmic action) と呼ばれる．それ以上分解不可能なプロセスの基本単位である．+は，選択演算子 (choice operator 又は alternative composition) あるいは和 (sum) と呼ばれる．$x + y$は，x 又は yの内どちらかを非決定的に選択して実行するプロセスを表わす．·は逐次合成 (sequencital composition) あるいは積 (product) と呼ばれ，$x \cdot y$は，先に x を実行し，その終了後に yを行うプロセスである．また，$x \cdot y$を xyと表記することもできる．

プロセス代数のまとめ

プロセス代数は，オブジェクト指向的な考えに加えて，観測可能性及び双模倣性，時間的関係を重要視しており，その最大の特徴は，単純な仕組みから導かれる形式性と汎用性にある．プロセス代数では，プロセスを「対象の行動パターンを記述する，周りの環境から観測されるイベント (情報の入出力の出現) の有限集合」として定義する．非常に単純な基本モデルに基づいており，一般性及び拡張性に富んでいる．すなわち，システムの要求仕様や設計仕様を，計算機処理が可能なように，形式体系に基づいて厳密に整った構造を持たせて記述する形式仕様記述言語による表現への活用が行える．また，通信プロトコル，システムインタフェース，開放型の並行分散システム，人間機械系，ソフトウェアプロセスといった複数の実行体系が互いに通信しあいながら並行的に動作するシステムや状況のモデル化，仕様記述，設計などに有効な形式体系への活用が可能である [二木 92]．

通信プロトコル記述言語の LOTOS [ISO89] [高橋 90a] [高橋 93] や SDL [CCI88] [若原 90] などは，プロセス代数の CCS 及び CSP に基づいて設計されている．

本論文では，このプロセス代数を用いることによって以降で確立する設計生産活動に関する基本モデル，その基本モデルに基づいた形式仕様記述言語の数学的な記述を行うことができると考えられる．また，形式仕様記述言語で記述した内容を，プロセス代数に基づいた数式表現に落して，プロセス代数の数式処理の範中で設計生産活動に関する活動記述の妥当性の判定や活動記述の簡約化などに活用することができると考えられる．

具体的な活用としては，ある作業で試行錯誤を繰り返しながらモデルを構築する作業を， CAD システムに対する入力列として表現し，その過程を簡約化し特定のモデルを構築する一般的なプロセスモデルとして表現すること，同じくモデルを構築する入力過程と，制約条件を規定する作業からパラメトリックモデルを構築するプロセスモデルなどを導出できる活用が考えられる．

3.2 本論文で提案する基本モデル

本節では，3.1.2で得られた課題を克服するために，3.1.3で述べたプロセス代数とオブジェクト指向を基にして，本論文で提案する設計生産活動を表現するための基本モデルの提案を行う．

3.2.1 基本モデルの概要

本論文では，設計生産のプロセスを「情報の入出力の過程」として表現することを基本モデルとして提案する．そのモデルに従って情報の入出力対象，及び入出力現象によって表現されるプロセスモデルを，一般的な設計及び生産のプロセスと区別するために設計生産コミュニケーションモデルと呼ぶ．

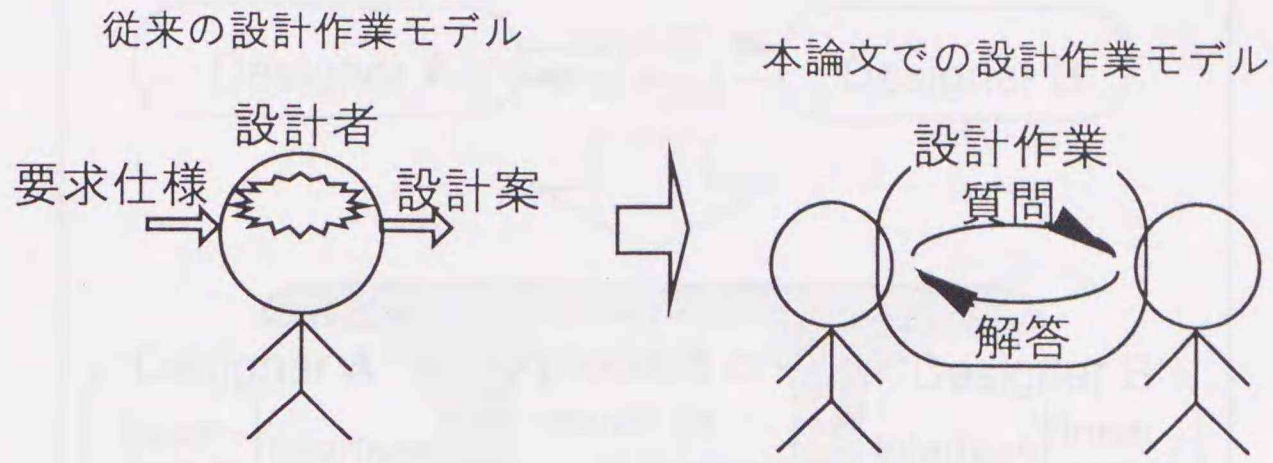


図 10: 情報入出力の過程をプロセスして扱う基本モデル

設計生産プロセスの基本モデル

「情報入出力の過程」をプロセスとする基本モデルに基づいて設計生産の過程を記録する方法として，情報の入出力過程を連続的な記録として残す方法がある．この方法は，１対１の情報の入出力では容易に扱えるが，複数の対象間で並行して情報の入出力が行われる状況では，モデルとしての意味を持たせることや，その後の活用を行うことが難しい．従って，一つ一つの入出力をプロセスとして個別に扱うのではなく，特定の活動に関する一連の情報の流れを取りまとめて，一括して取り扱えるように名前付けを行う必要がある．そこで本論文では，プロセスを単なる入出力の記録だけではなく，特定の活動が行われる間継続的に存在して情報を伝達する概念的な媒体としてモデル化する．図 10に，互いの通信の状況を集約してプロセスとして扱う概念を示す．

この形式では，ある特定の活動に関する情報は，すべて一つのプロセスを経由してやり取りされることになる．こうすることによって，全体としては複数の対象が並行して情報を交換している状況も，特定の活動を担うプロセスの視点から見ると，情報を順番に受け取り出力する形で記録することができる．

これまでは，プロセスの視点から見て一つの対象から情報が入力され，プロセスから反応が出力される例について述べてきた．しかし，実際の設計生産活動では一つの対象（例えば設計者）から複数のプロセスや対象（たとえば複数の設計グループなど）へ情報の入出力が行われる可能性がある．そこで，本論文では複数のプロセスや対象への同時に情報の出力，また同種の全てのプロセスや入出力対象に対して同時に同種の情報を出力の表現が必要と考えられる．

設計生産に関与する対象の基本モデル

本論文では，設計者や設計グループなどの人間及びその組織，CAD/CAM システムなどのアプリケーションプログラム，工作機械やロボットなどの製造設備の表現対象を，プロセス代数の見方に基づき「インタフェースが規定された入出力の対象」として扱う．すなわち，表現対象はブラックボックスとして，内部の活動を表現しないで扱える形式で表現する．またプロセスは，情報の連続した入出力現象であるが，特定の対象ごと，または特定の時間ごとにまとめることができる．すなわち，プロセスも特定の入出力対象から見ると，独立した情報入出力対象として扱うことができる．

設計活動を具体的な例として図 11に示す．設計者 A から B への情報の伝達は，その伝達の機会ごとにプロセスが生成されて消滅するのではなく， A から B への情報伝達の中で，ある意味をもつ

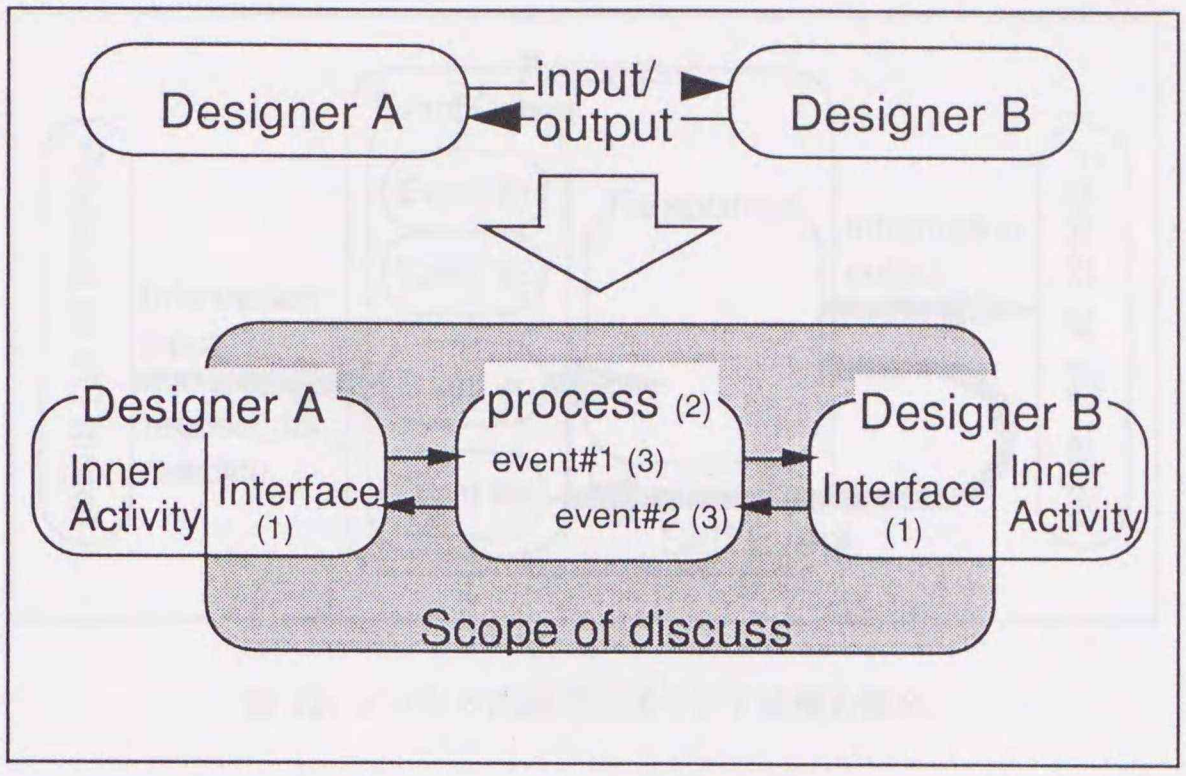


図 11: 設計活動を例とした入出力対象とプロセスの表現

情報の入出力作業が続く間，その作業に対応するプロセスが存在し，このプロセスを経由して情報伝達を行う形で表現する．その際に，各設計者を表現するインターフェース (1)，特定の作業における情報の伝達を行うプロセス (2)，その活動 (プロセス) においてある意味をもつイベントとして扱われる情報の入出力現象 (3) が，議論の対象となる．

また複数の設計者が異なった目的のために並行して作業している状況を図 12に示す．

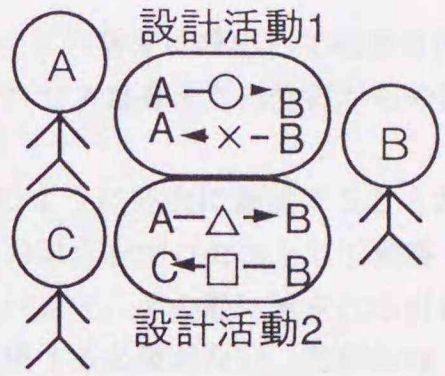


図 12: 設計活動を例とした並行活動プロセスの表現

この図では，複数の設計者 (ＡＢＣ) が複数の活動目的 (設計活動 1，2) に関して互いに並行して作業を進めつつ情報を交換している．このような状況は，その活動目的毎のプロセスが存在し，全ての情報をそのプロセスを経由することによって交換する．こうすることによって，全体としては複雑な活動も個別のプロセス毎に見ると，時系列的に情報の交換が行われているように見ることができる．

入出力対象とは，設計者や設計グループなどの人間及びその組織，CAD/CAM システムなどのアプリケーションプログラム，工作機械やロボットなどの製造設備を表現する通信対象であり，本論

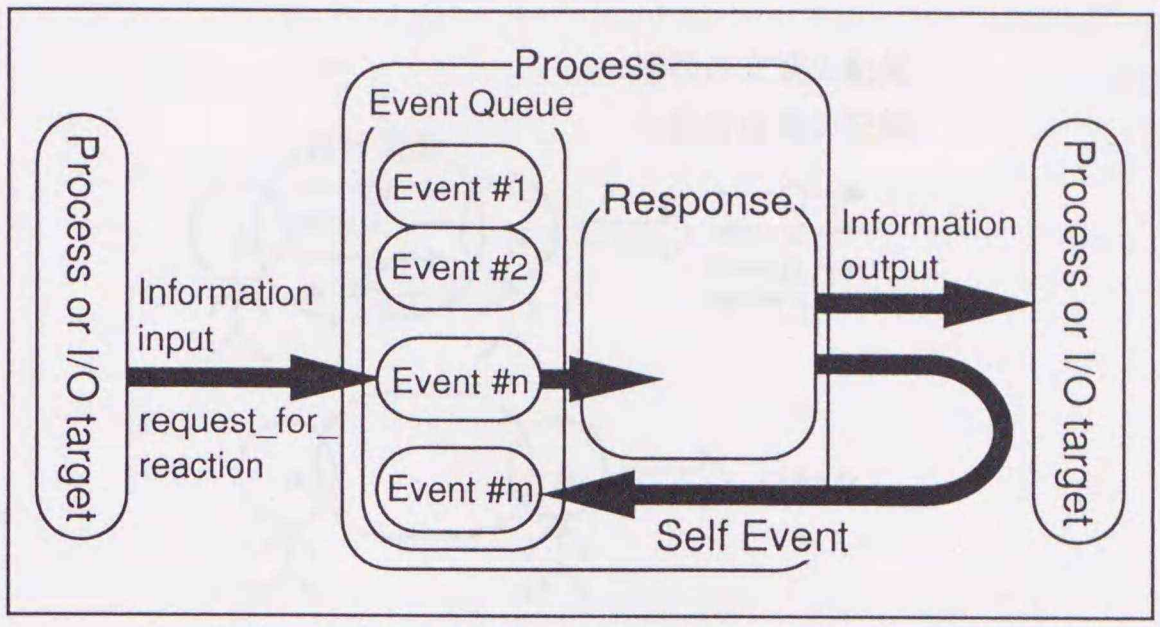


図 13: プロセス内部でのイベント処理の概念

文の考慮の対象とすべきものである．これらの対象に対しても情報の入力は行われるので，その入出力仕様は表現される必要がある．また入力された情報に対する対応を仕様とする必要があれば，そのアルゴリズムも表現できなくてはならない．

この時プロセスと入出力対象とは,「情報を入出力する相手」としての意味においては，区別する必要は無いと考えられる．従って，本論文ではプロセスと入出力対象の両方に関して， 基本モデルとしても 形式仕様記述言語の構文仕様上においても区別は行わず，入出力対象であっても内部処理のアルゴリズムを記述することを可能にする必要がある．

プロセスの内部動作表現

プロセスの内部動作とは，イベントの発生に呼応して起動されるプロセス内部の処理及び外部への情報の出力である．その動作はプロセス自身または外部からの情報の入力のイベントによって起動される．この関係を図 13に示す．

しかし例えば設計者の思考過程のように完全に表現することが不可能な場合もある．また本論文の本来の目的は，活動の中での情報の動きをプロセスとして表現することにある．この意味においては，入出力対象は入力されるメッセージと，その際に渡される引き数に関する仕様が表現されていれば良く，その内部動作は必ずしも表現する必要がない．内部処理のアルゴリズムを記述するかどうかは，どのようなモデルを表現するかどうかに依存する．従って本論文では，入力された情報の処理に関する記述は省略可能とした．プロセスも含めた入出力対象の内部処理アルゴリズムの記述表現が不可能であったり詳述する必要のない場合は，その部分の記述を省略する形で表現する．

情報伝達の制限や特別な処理，外部に対する情報の出力などが必要な場合は，そのプロセス内部で処理が行われる形で表現する．この形式ではプロセス内部の動作は，必然的に入力が行われる度に処理を行う形になる．最も単純な処理は，入力された情報を指定された相手にそのまま出力する形式である．また，モデリングを容易にするため，そのプロセスの内部処理の内で共通に行われる特定の処理を括り出して，自分自身に対して何等かの働きかけをすることで，その内部処理を起動させることも考えられる．本論文では，この内部処理を起動するきっかけをイベントと呼ぶ．先に述べたようにイベントには，自身の内部処理を起動するために自分で起動するものと，外部からの情報の入力と

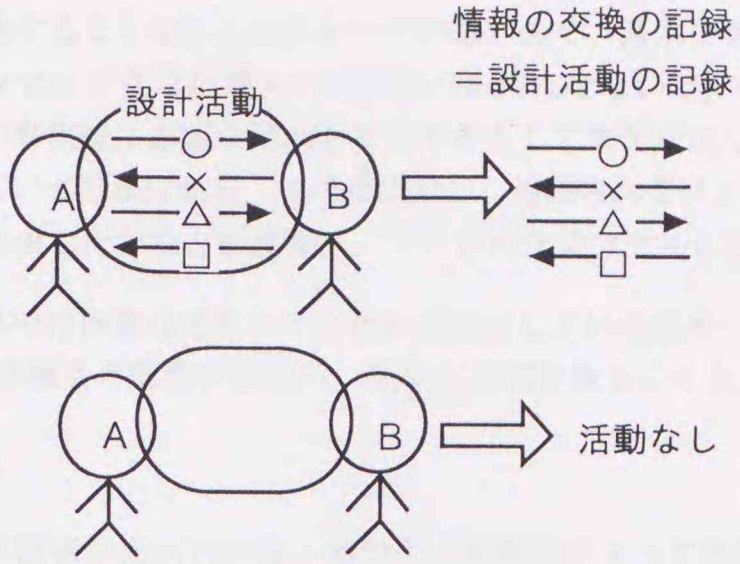


図 14: 基本モデルに基づく活動の記録

の 2 種類がある.

活動の記録

これまで述べてきた基本モデルに基づく，活動の記録について考える．本論文で提案した基本モデルでは，特定の活動に関与する各対象の内部状態や内部の挙動を記録することはできない．活動の記録として残されるものは，各対象の間での情報の交換のみでしかない．そこで，基本モデルに基づいた形での活動の記録は，その活動に対応するプロセスに注視した場合の，情報の入力と内部処理の記録となる．すなわち，イベント内の処理が確立的に変動する場合を除いて，プロセスに発生するイベントの時系列的な記録と，そのイベントが発生するごとのプロセスの内部変数 (プロセスの状態表現) の変化の記録とが，そのプロセスの記録となる．最終的には，その活動の記録は全てプロセス内部に関する情報で表現できることになる．図 14に活動の記録としての情報の交換と，情報交換が行われない場合の活動の記録の取り扱いを示す.

メッセージに対するアクションの排他的実行

並行した複数のプロセスを同時に実行している中で特定のメッセージ処理を行っている間に別のプロセス／入出力対象から情報を取り出す動作をする際に，次の問題が考えられるる.

- メッセージを受け取る毎に対応を行ってしまうので，その対応毎にプロセスの内部状態が変化してしまう．従って，プロセスの内部状態を保存することができない.
- 特定のメッセージを受け取った後で，他のプロセス／入出力対象からメッセージを待つ状況の表現が行えない.

このような問題に対しては，次のような対応が考えられる.

- あるメッセージを受け取って処理を行っている間は，その処理で必要としている特定のプロセスからの入力以外を受け付けない形で，排他的な規制をプロセスの定義として行えるようにする.

- 現実の世界での個々の設計者の活動に関して考えると，ある作業中に他からのメッセージの入力を自動的に拒絶するようなことは行われていないので，基本モデルや形式仕様記述言語の仕様としては，メッセージ受信に関する制限能力を持たせないで，モデル化の技術で対処する．すなわち，処理の単位毎に必要な情報をカプセルとしてモデル化し，プロセスに対する入力引き数として渡していく方法である．この形式では，処理されるメッセージの時系列的な変化や，どのプロセスを経由したかなどの経歴を，データの化カプセルに記録することが可能となる．

本研究では，実世界での作業などのモデル化を対象としているため，モデリングの容易性や計算機システム上での問題は残る可能性があるが，後者の方式を取ることとする．

基本モデルのまとめ

これまで述べてきた情報の入出力対象，及び入出力現象によって表現されるプロセスモデルを，一般的な設計及び生産のプロセスの概念と区別するために， 設計生産コミュニケーションモデルと呼ぶ．プロセス，入出力対象，情報の入力，イベントに関して，これまで述べてきた内容を箇条書きにしてまとめると次の項目になる．

- プロセス
  特定の活動を情報の流れに着目して表現するための概念的な情報伝達の媒体と考える．特定の活動毎に対応する媒体としてプロセスが存在し，その活動に関する全ての情報は，このプロセスを経由することによって伝達される．活動の記録は，プロセスに入力された情報と，その入力に対して実行された処理の履歴として表現される．入力された情報に対する何等かの処理と外部に対する反応をプロセスの動作として表現する．

- 入出力対象
  プロセスが実体としては存在しない情報の伝達を行う概念的な媒体と捉えられるのに対して，設計者や設計グループなどの人間及びその組織，CAD/CAM システムなどのアプリケーションプログラム，工作機械やロボットなどの製造設備などの，情報の入出力に関与する実世界の事物を入出力対象として表現する．情報の入出力の相手としては，入出力対象とプロセスは等価に扱われ，プロセスから他のプロセスや他の入出力対象へと情報の入出力を行うことができる．ただし入出力対象同士で直接情報を交換することはない．

- 情報の入力
  個別の情報の入力は，プロセスからはイベントとして処理される．個別の入力に対する反応などを含めた特定の活動に関する情報の流れが，プロセスの定義として表現される．

- イベント
  プロセスの内部処理を起動させるきっかけであり，プロセスの外部から観測することができるプロセス上の変化．プロセスや入出力対象の内部活動は，その外部からは観測されない．自身によって起こされるイベントと外部からの情報の入力の 2 種類がある．

- 活動の記録
  特定のプロセスに発生するイベントの時系列的な記録と，プロセスの内部変数 (プロセスの状態変数) の遷移の記録．プロセスの内部変数が変化することは，そのプロセスの内部活動を実行する際の条件が変化することであり，プロセス実行の状態が変化したことになる．

3.2.2 プロセス代数／オブジェクト指向に基づく定式化

定式化の必要性

本節では，提案した基本モデルが具体的にどのようなメカニズムで動作するか，に関して定式化を行う．

これは3.2.1で提案したプロセスを，例えば計算機支援システムの内部で実現することを考えた場合には，個別のプロセスの活動だけでなく情報の伝達やプロセスの生成などの機能の部分も具体的に実現されなければならないからである．なぜならば第4章で提案する形式仕様記述言語は，モデルを定義する人間が指定すべき情報のみを表現できればよく，システム全体としての整合性を保つ部分に関しては仕様化する必要はない．これらは，通常は実装時に開発者が決定する種類の問題だからである．又この部分の機能は当然ながら形式仕様記述言語を用いて表現することはできない．

従って，形式仕様記述言語の仕様の範囲内及び範囲外の部分も含めて，基本モデルとして提案したモデル全体の挙動をプロセス代数を基本として定式化する必要があり，その定式化された挙動のどの部分を言語仕様として実現するか，どの機能を計算機支援システムにおいて実装する必要があるか，を改めて検討する必要がある．

ＣＣＳによる定式化の必要性

以下では3.1.3で述べたCCSに基づいた形式で，説明だけでは十分に詳細化できていなかったプロセスと入出力対象及びシステム全体の挙動を表現する．

プロセス代数を用いた定式化を行う理由としては，オブジェクト指向(言語)を用いて定式化した場合，同時に並行して複数のメッセージが送られる処理を行う点の記述が繁雑化する恐れがある．またオブジェクト指向言語では，実装する環境に依存して表現の方式が変わる恐れがある．

このために，実装環境に依存せず，並行した動作を数学的に表現するためにプロセス代数に基づいた定式化を行う．

プロセス代数に基づく定式化

前節で提案した基本モデルは，基本的にはCCSに基づくものである．3.1.3の調査では，CCSによるプロセスの表現が通信における情報の入出力に基づくものであり，CSPは失敗によるデッドロックの概念に基づいたブラックボックスをイベントによる遷移システムとして構成し，これをプロセスとして扱う．別な言い方では，CCSは相互に接続された通信環境におけるシステムの挙動を仮定しており，CSPは特定のブラックスボックスの挙動を表現するための仕組みと考えることができる．本論文で対象としているプロセスは，不特定多数の対象によって同時に並行して実行される活動であり，特定のシステムを構築することを目的とはしていないので，通信の媒体としてプロセスを捉えるCCSの概念に基づいた形でのモデル化が適当であると考えられる．

CCSでのプロセスが持つポート及びポートにおけるアクションは，本論文においては情報(メッセージ)の到着，即ちイベントとして扱われる．識別子が$\#id$であるプロセスのポート$port_name$における引き数$args$を伴うイベントを$e^{\#id}_{port_name}(args)$としてCCSでの遷移システムとして表現する．タイプ$process_type$のプロセス全ての識別子が含まれる集合$ID^{process_type}$を考えると，個別の識別子$\#id$は集合に含まれるので，$\#id \in ID^{process_type}$となる．本論文におけるプロセスと入出力対象の挙動$C$は，再帰的に次の式で表現される．

$$C \stackrel{def}{=} e^c_i(x).\tau^c.out^c_j(y).C \tag{46}$$

ここで$e_1^c, e_2^c, \cdots, e_n^c$は，対象としているプロセスにおいて予め定義されたイベント，$out_1^c, out_2^c, \cdots, out_m^c$は対象としているプロセスから他のプロセス又は他の入出力対象へと発信されるイベント，x及びyはイベントにおける引数である．

すなわちプロセス (入出力対象) の動作は，何かイベントがあると内部動作が起動され，何等かの形で出力を行う形でモデル化する．入力に対する内部動作と対象への情報の出力がと終わった後は，初期状態に戻って次の入力に対して対処することになる．ただし，情報が入力されても内部状態が変化するだけで他への出力を行わないモデルも考えられる．従って$out_1^c, out_2^c, \cdots, out_m^c$には無動作のアクションも含まれることになる．

ＣＣＳによる定式化の問題点

ここで考えられる問題点として，CCS では，情報の入出力には必ず相補アクション名が必要になるが，本論文で提案しているプロセスからの情報の出力の相手は，常に確定している訳ではなく，外部からの情報によって選択的に変化しうる．すなわち相補アクション$out_1^c, out_2^c, \cdots, out_m^c$が，プロセスが実行されるまで確定されないことになる．又 CCS では，1 対多の入出力が規定できないことになっているが，本論文では活動中の同種の全てのプロセスに対して同じメッセージを発信することを必要としている．

この問題を解決するために，本論文では動作中の全てのプロセスや入出力対象を掌握し，あるプロセスからプロセス又は入出力対象への全ての情報の入出力を通過させるプロセスマネージャが常に活動しているとする．このプロセスマネージャは，全てのプロセスと入出力対象のポートを把握しておりメッセージ入力が可能になるように設定する．

各動作におけるプロセスマネージャと各プロセスの通信状況を図 15に示す．

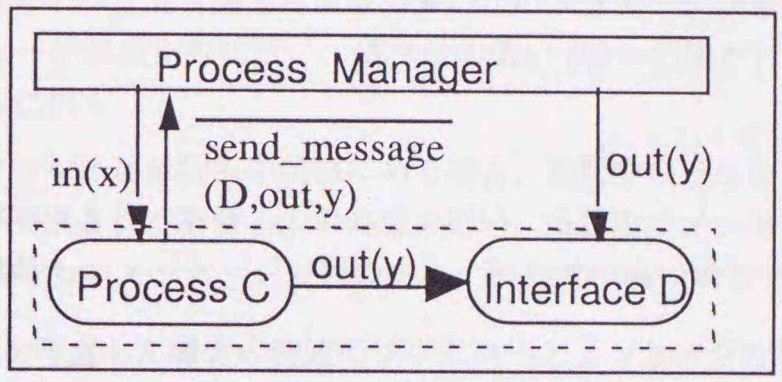


図 15: プロセスマネージャの概念

あるプロセスから他のプロセスや入出力対象に対しての情報発信は，プロセスマネージャに対して，実際の出力の相手と具体的なメッセージをカプセル化して渡す．すなわちプロセスマネージャのポートは，情報の入出力，プロセスの生成や終了など数種類しかない．他の全てのプロセス及び入出力対象からの情報の発信は，一旦このポートへと集められる．ポートにおけるアクションは，引数として渡された本来の出力相手のプロセスや入出力対象に対して，実際のメッセージを発信することである．プロセスマネージャは，動作中の全てのプロセスを把握しているので，対象とされるプロセスに対してのポートも全て把握している．この形式にすることで，上で述べた相補アクションに関する問題を解決することができる．

すなわちプロセスマネージャのアクションをリラベリングすることによって，通常のプロセスとプロセスマネージャとの間で発生するアクションを隠蔽し，プロセス同士の関係として表現することができる．

ＣＣＳによる定式化

以下では，他のプロセスに対してメッセージなどを発信する側のプロセスの識別子を#1 として例示しながら述べる．

1. メッセージ処理の優先度を実現するメッセージの受発信

   プロセスに対するインターフェースの定義として受け付けることが可能なメッセージを定義するが，このメッセージの処理の優先度を指定可能にする必要がある．例えば，ある設計作業を行っている最中に，その設計活動そのものが打切りになってしまった場合，その設計作業が終了するのを待たずに，設計活動そのものをただちに終了させる必要がある．すなわち，プロセスであるメッセージを受け取り処理を行っている最中に，他のプロセス／入出力対象からメッセージが届いた場合に優先度を比較して，より優先度の高いメッセージ処理をプロセスマネージャが優先させることが必要になる．

   この機能を実現するために，プロセスマネージャは，当該プロセスが現在処理中のメッセージの種類とその優先度を前もって把握しておき，別のメッセージが到着したら，次の場合分けで処理を行う．

   (a) 処理中のメッセージよりも優先させるべきメッセージが発信された場合，当該プロセスのメッセージ処理を中断し，優先度の高いメッセージ処理を先に行う．

   (b) 処理中のメッセージよりも優先させる必要のないメッセージが発信された場合，当該プロセスのメッセージ処理を先に行い，優先度の低い (後から来た) メッセージ処理を待ち行列に入れて次に行う．

   (c) 複数のメッセージ処理が待ちの状態にある場合，処理中のメッセージよりも優先度の高いメッセージが発信された場合 (a) の処理を行い，処理中のメッセージよりも優先度が低い場合は，処理待ちのメッセージの列の中に，優先度の高い順でメッセージを実行させる．

メッセージ優先度を考慮した場合の特定のプロセスへとメッセージを送る際のプロセス CとプロセスマネージャPMの動作は次の式で表される．

$$C = e^{\#1}_{ports}(x).\tau^{\#1}.e^{pm}_{message_pass}(\#j, ports, y).\{\ \} + e^{\#1}_{stopped}()\}.C \quad (47)$$

$$PM = e^{pm}_{message_pass}(\#j, ports, y).\tau^{pm}.\{\{\ \} + e^{\#j}_{stop}()\}.e^{\#j}_{ports}(y).PM + e^{pm}_{stopped}(c).\tau^{pm}.\{e^{\#j}_{ports}(y) + \{\ \}\}.PM \quad (48)$$

プロセス Cの動作の中で $e^{pm}_{background_ports}(\#j, ports, y)$ は，本来の命令を梱包して運ぶプロセスからプロセスマネージャへの出力であり，入力された各プロセスのイベントに対する相補アクションとなる．実際の命令の出力先のプロセスは$\#j$であり，そのポートは $ports$，ポートへ渡す引数は yとなる．プロセス Cは，他のプロセスに対してメッセージを発信した後，メッセージ処理が終わった場合には $e^{\#1}_{stopped}()$ をプロセスマネージャに対して発信して，メッセージの入力待ち状態になるか，メッセージ処理を続けることになる．プロセスマネージャは，$e^{\#1}_{stopped}()$ が発信されたかどうかで，メッセージ処理を続けているかどうかの判定を行う．

プロセスマネージャの動作τ^{pm} は，受け取ったメッセージの優先度をプロセスが処理中のメッセージと比較して，優先度に従って本来の通信相手にメッセージを送り出すか，現在処理中のメッセージを停止させて，新しいメッセージを送り返す．また，メッセージ処理の終了 $e^{\#1}_{stopped}()$ を受け取った場合は，メッセージの待ち行列がある場合には，次のメッセージを当該プロセスに対して発信する．処理すべきメッセージが待ち行列に存在しない場合は何もしない．

2. 複数プロセスへの入力
複数のプロセス又は入出力対象に対する情報の入力の場合は，上のモデルでの対象となる識別子$\#j$が集合 $(\#l, ..., \#m)$ に拡張される．この場合はメッセージを受け取る個別のプロセスで優先度の高いメッセージ処理を行っているかどうかが問題となり，個別のプロセスのメッセージ処理状況を見て，必要に応じて割り込みをかけるか，優先度の高いメッセージ処理を待つ必要がある．数式表現ではプロセス Cからのメッセージ発信の部分が拡張されて次の式になる．

$$C = e^{\#1}_{ports}(x).\tau^{c}.e^{pm}_{message_set}((\#l, ..., \#m), ports, y).\{\{\} + e^{\#1}_{stopped}()\}.C \quad (49)$$

$$PM = e^{pm}_{message_set}((\#l, ..., \#m), ports, y).\tau^{pm}. \{e^{pm}_{message_pass}(\#l, ports, y) \ldots e^{pm}_{message_pass}(\#m, ports, y)\}.PM \quad (50)$$

プロセスマネージャは，発信されたメッセージを受けとり，指定された集合の各要素に対してのメッセージ発信に分解して，自身に対して改めてメッセージを発進の要求を出す．この時のプロセスマネージャの内部アクションτ^{pm}は，与えられたメッセージを送り出す対象の集合の各要素に対して，メッセージ発信の要求を送り出すことである．個別のプロセス毎のメッセージ優先度の処理は，前項目に従って行う．$e^{pm}_{stopped}(c)$ を受け取った際の動作は，式 (48) と同じなのでここでは省略している．

3. 同一種類の全プロセスへの同一メッセージの発信
活動中の同種の全てのプロセスに対する同報機能の実現は，上の例での集合 $(\#l, ..., \#m)$ がより一般的なプロセスのタイプ指定 $process_type$ へと拡張されたと考えられる．メッセージの優先度の処理や他の部分は同じで表現でき，数式表現では次の式となる．

$$C = e^{\#1}_{ports}(x).\tau^{c}.e^{pm}_{message_set}(process_type, ports, y).\{\{\} + e^{\#1}_{stopped}()\}.C \quad (51)$$

$$PM = e^{pm}_{message_set}(process_type, ports, y).\tau^{pm}. \{e^{pm}_{message_pass}(\#1, ports, y) \ldots e^{pm}_{message_pass}(\#n, ports, y)\}.PM \quad (52)$$

プロセスマネージャは，発信されたメッセージを受けとり，自身で管理している活動中のプロセスの集合から指定された型の活動中のプロセスの集合$\#id \in ID^{process_type}$ を作り出し，上記の複数プロセスへの発信と同じ要領で，自身に対して集合の各要素を引き数としたメッセージ発信の依頼を出す．この時のプロセスマネージャの内部アクションτ^{pm}は，指定された型の活動中の全てのプロセスを網羅した集合を作り出すこと，メッセージを送り出す対象の集合の各要素に対して，メッセージ発信の要求を送り出すことである．

$e^{pm}_{stopped}(c)$ を受け取った際の動作は，式 (48) と同じなのでここでは省略している．

4. プロセスの生成
このプロセスマネージャの概念は，新たなプロセスの生成や消滅を行う際も活用することができる．なぜならば，全てのプロセスや入出力対象に対するポートを把握するためには，プロセ

スや入出力対象が，勝手に出現するのではなく，プロセスや入出力対象とこのプロセスマネージャとがプロセスや入出力対象の生成に関与する必要があるからである．

プロセスの生成は，メッセージを発進しようとしているプロセスから，プロセスマネージャに対して，起動される対象又はプロセスの型が送られる．数式表現では

$$C = e^{\#1}_{ports}(x).\tau^{c}.e^{pm}_{start_process}(process_type, \#1).\{\{\} + e^{\#1}_{stopped}()\}.C \quad (53)$$

$$PM = e^{pm}_{start_process}(process_type, \#1).\tau^{pm}.e^{\#1}_{started}(\#id).PM \quad (54)$$

となる．プロセスマネージャは，発信されたメッセージを受けとり，指定された型のプロセスを起動して同種のプロセスの中で一意な識別子を割り当て，その型の活動中プロセスの集合に加えて，作り出したプロセスの識別子をプロセスの生成を要求したプロセスに対して返す．従って，この時のプロセスマネージャの内部アクションτ^{pm}は，プロセスの起動，識別子の割り当て，指定された型の活動中の全てのプロセスを網羅した集合の操作である．

$$\#id \notin ID^{process_type} \quad (55)$$

$$ID^{process_type} = ID^{process_type} + \#id \quad (56)$$

相補アクションは，プロセスの生成を要求したプロセスに対して生成したプロセスの識別を返すことである．

5. プロセスの終了・消滅
   プロセスの終了及び消滅には，2 種類の場合が考えられる．1 つは，プロセス自身が終了条件を満たして停止する場合．もう一つは，他のプロセスから強制的に終了させられる場合である．

   プロセスが自身の終了条件を満たして終了する場合は，そのプロセスから，プロセスマネージャに対して，終了を示すメッセージが送られる．数式表現では次の形式となる．

$$C = e^{\#1}_{ports}(x).\tau^{c}.e^{pm}_{stop}(\#1).\{\{\} + e^{\#1}_{stopped}()\}.C \quad (57)$$

$$= e^{\#1}_{halt}().\tau^{c}.0 \quad (58)$$

$$PM = e^{pm}_{stop}(\#1).\tau^{pm}.e^{\#1}_{halt}().PM \quad (59)$$

   プロセスマネージャは，発信されたメッセージを受けとり，終了するプロセスと同種のプロセスの識別子の集合から，停止するプロセスの識別子を取り除き，メッセージを発信したプロセスを，停止して消滅させてしまう．

$$\#1 \in ID^{process_type} \quad (60)$$

$$ID^{process_type} = ID^{process_type} - \#1 \quad (61)$$

   従って，この時のプロセスマネージャの内部アクションτ^{pm}は，指定された型の活動中の全てのプロセスを網羅した集合から，終了するプロセスの識別子を取り除く操作と実際のプロセスの消滅だけである．この場合，相補アクションは特に必要が無い．

6. 他のプロセスからの別のプロセスの停止
   プロセスが他のプロセスから強制的にする場合は，終了メッセージの発信と停止させられるプロセスが異なっている点だけが，上の記述と異なっている．メッセージは，終了させようとし

ているプロセスから，プロセスマネージャに対して，他のプロセスの終了要求するメッセージが送られる．数式表現では

$$C^{\#1} = e^{\#1}_{ports}(x).\tau^{c}.e^{pm}_{halt}(\#1,\#n).\{\ \{\ \} + e^{\#1}_{stopped}()\}.C^{\#1} \quad (62)$$

$$C^{\#n} = e^{\#n}_{halt}().\tau^{c}.0 \quad (63)$$

$$PM = e^{pm}_{halt}(\#1,\#n).\tau^{pm}.e^{\#n}_{halt}().e^{\#1}_{halted}(\#n).PM \quad (64)$$

となる．プロセスマネージャは，発信されたメッセージを受けとり，終了するプロセスと同種のプロセスの識別子の集合から，停止するプロセスの識別子を取り除く．

$$\#n \in ID^{process_type} \quad (65)$$

$$ID^{process_type} = ID^{process_type} - \#n \quad (66)$$

従って，この時のプロセスマネージャの内部アクションτ^{pm}は，指定された型の活動中の全てのプロセスを網羅した集合から終了するプロセスの識別子を取り除く操作だけである．この場合，相補アクションはプロセスの終了を要求したプロセスに対してプロセス終了の確認を送ることである．現実のモデル化では，他のプロセスから任意のプロセスを終了させることは，モデルの安定性上問題があるので，この問題に関する制約条件を定義できるようにする必要がある．

7. プロセスの活動の確認

プロセスの生成と終了とに，プロセスマネージャを用いた形式にすることによって，あるプロセスから他のプロセスの動作の確認を行うことが可能になる．

即ち，全てのプロセスは並行に活動しているので，あるプロセスで対象とするプロセスの識別子を内部変数として保持していても，相手のプロセスがいつまで動作していて，その識別子が有効であるかは全く確定されないことである．また対象としているプロセスに対してメッセージを発信して，帰ってくるかどうかで判断するとしても，プロセスは外部観測によってしか，その内容を推定することができないので，返事が帰ってこない場合に，そのプロセスが終了して消滅してしまっているのか，終了していないが何等かの理由で返答が帰ってこないのか推定できないことになる．このような事態に対処するために，プロセスはプロセスマネージャに対して自身が保持している識別子を持ったプロセスが，その時点でも活動しているかどうかを問い質すことを可能にする必要がある．このメカニズムも，これまで述べてきた内容とほぼ同等に記述できる．

プロセスの活動の確認は，他のプロセスの活動を確認しようとしているプロセスから，プロセスマネージャに対して，確認する対象のプロセスの識別子と活動確認のメッセージが送られる．数式表現では次の式となる．

$$C = e^{\#1}_{ports}(x).\tau^{c}.e^{pm}_{is_alive}(\#1,\#n,process_type).\{\ \{\ \} + e^{\#1}_{stopped}()\}.C \quad (67)$$

$$PM = e^{pm}_{is_alive}(\#1,\#n,process_type).\tau^{pm}.e^{\#1}_{alive}(result).PM \quad (68)$$

プロセスマネージャは，発信されたメッセージを受けとり，指定された識別子のプロセスが活動中プロセスの集合にあるかどうかを確認して，確認を要求したプロセスに対して返す．

$$result = \begin{cases} True & (\#n \in ID^{process_type}) \\ False & (\#n \notin ID^{process_type}) \end{cases} \quad (69)$$

従って，この時のプロセスマネージャの内部アクションτ^{pm}は，指定された型の活動中の全てのプロセスを網羅した集合に当該プロセスの識別子が存在するかどうかの確認である．相補アクションは，プロセスの確認を要求したプロセスに対して真または偽の値を返すことである．

ここで，同じ識別子を複数回使った場合には，実際には別のプロセスが同じ識別子を使うことになる．この場合，実際には終了している筈のプロセスに対して活動中と判断されることになってしまう．従って，プロセスの識別子の一意性は，特定の時点のみならずプロセスマネージャが管理する時間全体において同じ識別子を用いないようにする必要がある．

8. メッセージ処理中かどうかの確認
前節と同じメカニズムで，他のプロセスでメッセージを処理中かどうかを判定することも可能になる．

すなわち，プロセスマネージャでは当該プロセスがメッセージを受け取って処理中であるか，待ち行列にメッセージが残っているかを把握できるので，その結果を送り返すことができる．

プロセスの活動の確認は，他のプロセスの活動を確認しようとしているプロセスから，プロセスマネージャに対して，確認する対象のプロセスの識別子と活動確認のメッセージが送られる．数式表現では次の式となる．

$$C = e^{\#1}_{ports}(x).\tau^{c}.e^{pm}_{is_running}(\#1,\#n,process_type).\{\{\} + e^{\#1}_{stopped}()\}.C \quad (70)$$

$$PM = e^{pm}_{is_running}(\#1,\#n,process_type).\tau^{pm}.e^{\#1}_{running}(result).PM \quad (71)$$

プロセスマネージャは，発信されたメッセージを受けとり，指定された識別子のプロセスが活動中プロセスの集合にあるかどうかを確認して，確認を要求したプロセスに対して返す．

ここまで述べてきたプロセスマネージャの機能 (又は内部動作τ^{pm}) についてまとめると次の項目になる．

1. 他のプロセス向けのメッセージと対象プロセスが渡された場合，当該プロセスが処理中のメッセージの優先度と，発信されるメッセージの優先度を比較して，処理中のメッセージの優先度が高い場合は，発信されるメッセージを待ち行列に入れて処理が終わった後に当該プロセスに渡す．処理中のメッセージよりも優先度が高い場合には処理を中断して，新しいメッセージを当該プロセスに渡し，中断したメッセージは待ち行列に入れる．本来の出力相手のプロセスや入出力対象に対して，渡されたメッセージを再び発信する．

2. 複数の対象を含んだ集合と，そのプロセス向けメッセージが指定された場合，指定された集合の各要素に対して渡されたメッセージを再び発信する．

3. 特定のタイプのプロセスのタイプと，そのタイプ向けのメッセージが指定された場合，指定されたタイプの活動中のプロセスの集合の全要素に対して実際のメッセージを発進する．

4. 特定のプロセスのタイプと，そのプロセスの生成のメッセージが入力された場合，指定されたタイプのプロセスを起動し，活動中のプロセスの集合から一意な識別子を起動されたプロセスに与え，そのプロセスの識別子を起動を要請したプロセスに返す．

5. 特定のプロセスから，そのプロセスの終了のメッセージが入力された場合，終了メッセージを発したプロセスの識別子を活動中のプロセスの集合から取り除いて，当該プロセスを消滅させる．

6. プロセスから，他のプロセスの終了のメッセージが入力された場合，終了メッセージを発したプロセスの識別子を活動中のプロセスの集合から取り除いて，当該プロセスを消滅させる．

7. プロセスから，他のプロセス活動中かどうかの確認メッセージが入力された場合，活動中のプロセスの集合に属しているかを確認し，真又は偽の結果を，要求を出したプロセスに返す．

8. プロセスから，他のプロセスメッセージ処理中かどうかの確認メッセージが入力された場合，メッセージを受け取った後で $e^{pm}_{stopped}(c)$ を PM に対して送ったかどうかで真又は偽の結果を，要求を出したプロセスに返す．

これまでのプロセスの動作及びプロセスマネージャの動作式は，各イベント毎でしか表現していないが，実際の動作では起こったイベントに対して反応を取ることになるので，全ての動作式は *or* (即ち CCS では + 記号) で結ばれる形になる．ここまで述べてきた全てのプロセス動作式を一つにまとめると式 (72) 及び式 (73) になる．

$$
\begin{aligned}
C \;=\;& e^{c}_{ports}(x).\tau^{c}.e^{pm}_{message_pass}(\#j, ports, y).\{\,\{\,\} + e^{\#1}_{stopped}()\}.C \\
+\;& e^{c}_{ports}(x).\tau^{c}.e^{pm}_{message_set}((\#l, ..., \#m), ports, y).\{\,\{\,\} + e^{\#1}_{stopped}()\}.C \\
+\;& e^{c}_{ports}(x).\tau^{c}.e^{pm}_{message_set}(process_type, ports, y).\{\,\{\,\} + e^{\#1}_{stopped}()\}.C \\
+\;& e^{c}_{ports}(x).\tau^{c}.e^{pm}_{start_process}(process_type, c).\{\,\{\,\} + e^{\#1}_{stopped}()\}.C \\
+\;& e^{c}_{ports}(x).\tau^{c}.e^{pm}_{stop}(c).e^{c}_{halt}().\tau^{c}.0 \\
+\;& e^{c}_{ports}(x).\tau^{c}.e^{pm}_{stop}(c, \#n).\{\,\{\,\} + e^{\#1}_{stopped}()\}.C \\
+\;& e^{c}_{halt}().\tau^{c}.0 \\
+\;& e^{c}_{ports}(x).\tau^{c}.e^{pm}_{is_alive}(c, \#n, process_type).\{\,\{\,\} + e^{\#1}_{stopped}()\}.C \\
+\;& e^{c}_{ports}(x).\tau^{c}.e^{pm}_{is_running}(c, \#n, process_type).\{\,\{\,\} + e^{\#1}_{stopped}()\}.C
\end{aligned}
\tag{72}
$$

$$
\begin{aligned}
PM \;=\;& e^{pm}_{message_pass}(\#j, ports, y).\tau^{pm}.\{\{\,\} + e^{\#j}_{stop}()\}.e^{\#j}_{ports}(y).PM \\
+\;& e^{pm}_{stopped}(c).\tau^{pm}.\{e^{\#j}_{ports}(y) + \{\,\}\}.PM \\
+\;& e^{pm}_{message_set}((\#l, ..., \#m), ports, y).\tau^{pm}. \\
& \{e^{pm}_{message_pass}(\#l, ports, y) \ldots e^{pm}_{message_pass}(\#m, ports, y)\}.PM \\
+\;& e^{pm}_{message_set}(process_type, ports, y).\tau^{pm}. \\
& \{e^{pm}_{message_pass}(\#1, ports, y) \ldots . e^{pm}_{message_pass}(\#n, ports, y)).PM \\
+\;& e^{pm}_{start_process}(process_type, \#1).\tau^{pm}.e^{\#1}_{started}(\#id).PM \\
+\;& e^{pm}_{stop}(\#1).\tau^{pm}.e^{\#1}_{halt}().PM \\
+\;& e^{pm}_{halt}(\#1, \#n).\tau^{pm}.e^{\#n}_{halt}().e^{\#1}_{halted}(\#n).PM \\
+\;& e^{pm}_{is_alive}(\#1, \#n, process_type).\tau^{pm}.e^{\#1}_{alive}(result).PM \\
+\;& e^{pm}_{is_running}(\#1, \#n, process_type).\tau^{pm}.e^{\#1}_{running}(result).PM
\end{aligned}
\tag{73}
$$

$$
result \;=\; \begin{cases} True & (\#n \in ID^{process_type}) \\ False & (\#n \notin ID^{process_type}) \end{cases}
\tag{74}
$$

上の式 (73) を見ればわかるように，プロセスマネージャ PM の動作式では，0 で終わる部分が無いので，プロセスマネージャが停止することは無い．

ここまで述べてきた，プロセスマネージャの動作は，形式仕様記述言語の仕様の範囲外になり，実際のモデルを構築する際には，それらの機能を意識して表現する必要はない．しかし，第 6章で行う計算機支援システムの開発においては，記述表現された各プロセスの機能を実現するだけでなく，各プロセスの活動を管理し，CCS としての一貫性を保たせるためにプロセスマネージャの機能を実現する必要があり，第 6章での重要な課題になると考えられる．

オブジェクト指向的な見方

オブジェクト指向的な見方では，本論文で提案した入出力対象及びプロセスは，それぞれクラスとして見ることができる．ただし，これらの 2 つのクラスを別々に表現することはできないので，抽象的な上位型として,「抽象的な入出力対象」を上位型として定義し,「プロセス」と「入出力対象」とをその下位型として定義する．

「抽象的な入出力対象」は，最も汎化された対象であり情報の入出力と，内部活動の表現に関するにおける特性，実体化されると独立して並行に活動を始める特性を定義する．各クラスのメソッドは，そのまま情報の入力として扱う.「プロセス」は,「抽象的な入出力対象」の下位型として全ての特性を継承し，実世界に存在しないで情報の入出力を制御する概念的な媒体としてモデル化される．

本論文で提案した概念での「入出力対象」は,「抽象的な入出力対象」の下位型としてその特性を継承し実世界に存在する対象である点と必要に応じては内部活動を表現しない部分での拡張がなされる．

また，プロセス代数として見たモデルで述べた，全てのプロセスの活動を把握し，個別のプロセスや入出力対象に対数情報の入出力を実行するプロセスは，マネージャクラスとして，概念的なプロセスクラスの特殊な下位型として扱われる．

実際にモデル化を行う際には，これらのクラス階層的な考えを進めて入出力対象とプロセスの下位型として具体的なモデルを表示形式仕様記述言語の仕様を確定する必要がある．この考え方は，第 6章で計算機言語による実装を考慮する際の有効であると考えられる．

挙動規定法から見た分類

また，本論文で提案したプロセスモデルは，4.1.4で述べる外部挙動規定法に相当する．これは，実装を特に意識せずに，通信を行う者同士が互いにやり取りする情報の順序関係だけを規定する方法で，状態 S を陽に記述せず，入力 I と出力 O の組み合わせにより有限状態機械を定義する方法に該当する．これは LOTOS[ISO89] [高橋 90a] が用いているのと同じ形式である．

3.3 並列処理言語としての機能比較

3.4 特徴及び他のモデルとの比較

3.4.1 本論文における基本モデルの特徴

本論文における基本モデルの特徴を，3.1.2でまとめた問題点と比較しながら述べる．

- プロセスに関する概念
  3.1.2でまとめた問題点では，SADT 手法的な箱の入れ子構造ではなく動的にプロセスの活動

表現を行う必要や，その中でプロセスを陽に表現して識別し，時間的な特性，他の活動との並行活動や時間的な前後関係，親子関係，複数の活動の協調などを記述することが必要とされた．本論文では，情報の入出力は並列して行われており，特定の活動を表現するプロセスに注目した場合には，情報は順序だてて入力されてイベントとして処理される．このような概念に基づいた本論文のモデルでの活動表現では次のような特徴を持つ．

- 複数のプロセスが並列して動作し，それぞれ独立して情報の入出力行うことによって並列活動を表することができる．また，プロセス及び情報の入出力対象それぞれでの情報の入出力は，特定の活動に関して順序だてて表現されることになる．
- 並行動作するプロセス同士の時間的相互関係は，特定のプロセスが次のプロセスを起動する際の時間的な前後関係として表現できる．また，起動されたプロセスが同期を取るためには，情報の入力を同時に行って起動されるアルゴリズムを同期させる方式が考えられる．このためには，次章で述べる 形式仕様記述言語の仕様に，時間的な情報取り扱える関数などの機能を設ける必要があると考えられる．

• 全体的な構成について

3.1.2でまとめた問題点の一つとして，特定のモデルに依存することによって，他の形式のモデルへの活用が難しくなる点を上げた．本論文で提案した基本モデルは，入出力の現象の観測をプロセスとして扱い，また入出力の仕様のみによって対象を表現している．

- この形式は基本的には並列プロセスモデルであることから，CIMOSA あるいはEXPRESS-P で表現される形式の並列プロセスモデルや離散的イベントシミュレーションモデルを表現することは可能である．
- またペトリネットのモデルでは，アークの構造が時系列的には変化しない有限状態機械のモデルである範囲では，プレース及びトランジションに相当するプロセスを置くことで等価なモデルを表現することが可能と考えられる．
- またメッセージを送って入力されたメッセージごとに処理を行う形式であるので，オブジェクト指向モデルの拡張と捉えることが可能なので，オブジェクト指向言語を用いて表現されたモデルを本論文で提案したモデルを用いて表現することも可能であると考えられる．
- プロセスを階層的に動作させて，特定のプロセスと内部のプロセスとが情報を入力する形式にすることで SADT 的な階層化された設計生産過程の構造化は，活動の親子関係などモデルを表現することで表現できる．
- ルールベースシステムは，一階述語論理に基づいて，制約条件を複数定義しこの制約を満たす解を探索するシステムである．本論文で提案したモデルは，特定の活動に関連した情報の動きを表現するためのものであるので，ルールベースシステムで表現しているモデルを表現することは難しいと考えられる．この点については，3.4.3においてまとめる．

• 対象の表現能力

3.1.2でまとめた問題点では，できるだけ広い範囲の対象を表現することの重要性を指摘した．

本論文で提案した手法では，入出力の仕様のみによって対象を表現する非常に単純な形式を用いている．このために各活動モデルの中で用いられる対象の表現を単純化することができる．こ

の方式を用いることによって設計生産活動に関係する設計活動者と設計グループ，CAD/CAMシステムなどのアプリケーションプログラム，NC 工作機械，ロボットなどの生産設備などの幅広い対象，及び各活動において扱われる情報，例えばSTEP の製品モデルなどをも入出力対象として表現できる広い表現能力を備えることができる．

- プロセス及び対象間での入出力
  3.1.2でまとめた問題点では，入出力する対象の種類やインスタンスを予め固定しておくことなしに，対象の構成が動的に変化しても情報を入力できる事．及び入力する情報の種類も，単純なシグナル的な表現ではなく，より高度な意味を持つ情報の入出力として行う必要性を指摘した．

  – 本論文のモデルでは，情報を入力する相手とのリンクを予め定義する事無しに相手の識別子を指定することで，情報を入力することが可能になる仕様にする．
  – また入力されるデータの種類は，予め定められたシグナルの出力やレジスタ／カウンタ値の変更などの低レベルではなく，属性を持った高度な情報を入力できるようにする．

  この 2 点に関しては，次の第 4章での形式仕様記述言語の仕様として実現する必要がある．

- プロセスの実行メカニズム
  本論文で提案した基本モデルは，基本的にはプロセス代数の CCS と，この理論に基づいた並列プログラミング言語を用いることによって実現することが可能であると考えられる．

3.4.2 他のモデルとの比較

本節では，3.1.1で調査した研究に対する評価を 3.1.2でまとめた問題点に注目してまとめる．

比較する項目は，3.1.2で上げられた 1) 動的なプロセス表現が可能であるか，2) 全体構成，3) 対象の表現能力，4) プロセス及び対象間での入出力の記述能力，とした．比較対象の研究として，1) Petrinet，2) FDL，3) SADT ，4) CIMOSA，5) COSL，6) 本論文で提案した設計生産コミュニケーションモデルに関して行った．

1. ペトリネットでは，表現されたモデルがペトリネットで表現できる範囲に限定されてしまう可能性が非常に大きい．この点は，ルールベースシステムに基づいた FDL についても同じことが言える．3.1.2で上げられた問題点に関して，この 2 つの研究は殆ど同じ特徴を示す．ペトリネットは表現する対象をプレース又はトランジションとすることで，いろいろな対象に対応させることができるが完全なものではない．また，トークンを複数用いる事による並行動作表現は行えるが，プロセスを陽に表現して一意に識別できている訳ではないので不完全と考えられる．

2. ルールベースシステムの FDL は，1.3.1で挙げた目的とは異なった目的のために用いられるものなので，特定モデルへの依存性，動的なプロセスの概念，入出力対象の表現，入出力データの表現のいずれにおいても満足できるものではない．

3. SADT 手法 (IDEF 等) は，特定モデルへの依存性ではプロセス構造の入れ子構造になる点が問題となる．すなわち SADT 手法では，プロセスを表現する際に特定プロセスの一意性やプロセスの相互関係の表現などが行えないことが起こる．入出力対象の表現は，1) や 2) の 2 つに比べると表現能力があると考えられる．

Subject	Petrinet	FDL	SADT	CIMOSA	COSL	DMCM
Dependency to specific model	×	×	△	○	△	○
Dynamic process concept	△	×	×	○	×	○
Representation ability	△	×	△	○	×	○
Representation of input/output	×	×	○	○	×	○

表 6: 活動表現の観点から見た既存の研究及び本論文の評価
○ : Completely support, △ : Not completely, × : Not support

4. CIMOSA 本論文とは，ほぼ同じ目的で，本論文の方が設計活動などを対象にしている分だけ広いと言える．CIMOSA で用いられている MMS 言語は，特定モデルへの依存性，動的なプロセス表現，対象表現能力，入出力データの表現に関してほぼ満足できる記述能力を持っている．ただし，入出力データの表現などの仕様を多機能にしている分だけ，表現されるモデルが複雑化されることが懸念される．

5. COSL は，デバイスレベルの活動表現を目的としているので，特定モデルへの依存性に関しては問題が少ないと思われるが，動的なプロセス表現や，入出力対象の表現，入出力データの表現に関しては，表現能力が足りないと考えられる．

6. 本論文で提案した，設計生産コミュニケーションモデル (DMCM) では，特定モデルへの依存性は，最低限に押さえて種々のモデルでも表現できることを目的としている．また，動的なプロセスを表現することを可能にし，設計生産に関与する対象の表現も，それらを入出力対象として扱うことによって多様な対象を表現する能力を持っている．入出力されるデータは，単なるシグナルレベルの内容でなく，より高度なレベルのデータを表現して入出力することが可能である．

ここまで述べてきた内容を表 6にまとめる．3.4.1で述べたように，本論文で提案した設計生産コミュニケーションモデルではこれらの問題点に関しては克服することができたと言える．

3.4.3 本論文での表現の限界

本節では，本論文で提案した設計生産コミュニケーションモデル での表現の限界に関して考察する．

- 本論文で提案したモデルでは，表現する対象を情報入出力の対象として表現する．この形式で表現できる範囲であれば問題はないが，逆に実際の搬送など物が移動したり変化するようなモデルでは，直接表現することは難しいので，何等かの形で情報入出力の形式に変換して表現する必要がある．又ルールベースシステムに基づいて表現されたモデルは，元にしている数学モデルが異なっているために，本論文での基本モデルに基づいて表現することは難しいと考えられる．

- プロセスを実際に動作する数だけ複数定義して並行活動を表現するのでは，表現されたモデルが複雑化してしまう懸念がある．この点は，第5章で述べるモデルの基本構造の部分で対応する必要がある．

- 入出力の構造が動的に変化するために，ペトリネットのように有限状態機械として安定性の数学的な判定を行うことは難しい．

また，次の項目は問題点ではないが，基本モデルの範中外で，次章で述べる形式仕様記述言語の仕様として定義する必要がある項目である．

- 時間的な情報取り扱える関数などの機能を設ける必要がある．

- 本論文のモデルでは，情報を入力する相手とのリンクを予め定義する事無しに相手の識別子を指定することで，情報を入力することが可能になる仕様にする．

- また入力されるデータの種類は，予め定められたシグナルの出力やレジスタ／カウンタ値の変更などの低レベルではなく，属性を持った高度な情報を入力できるようにする．

3.5 結 論

第3章では，はじめに設計生産の活動を表現するため，既存の研究に関する分析によって問題点の抽出を行った．得られた問題点を解消するために設計生産活動を表現するための基本モデルの提案を行い，以下の結論を得た．

- 設計生産活動を表現する既存の研究での概念に関する分析を行い，これまで用いられてきた方法論の分析を行った．

- 本論文で目的としている設計生産活動を表現するためには，特定のモデルに対する依存性が少ないこと，動的な活動を表現する必要があること，多様な対象を表現する必要があること，高レベルの情報を交換する必要があること，プロセスの実行メカニズム考慮した設計にする必要があることなどが判明した．

- 設計生産活動の表現以外の分野での研究として，オブジェクト指向，プロセス代数及び通信プロトコル記述言語などの研究が行われており，メッセージを受信することでメソッドが起動されるメカニズムや，プロセスを「周囲のプロセスや周りの環境と通信を行う抽象的なエージェント」としてとらえる考え方，プロセス代数における情報の入出力の観察結果をプロセスとして扱う概念が本論文の目的に適していることを明らかにした．

- 設計生産に関する活動と，その活動に関与する各種の対象を表現するための基本モデルの提案を行った．これはオブジェクト指向的な考えに加えて，情報入出力の観測結果をプロセスとして扱うプロセス代数の考え方を元にしたものである．

  - 設計生産における活動に関与する様々な対象を，その内部活動は表現せずに情報入出力の対象として表現する．
  - 設計生産におけるプロセスを，上記の情報入出力対象の間で，特定の活動に関する情報をやり取りする概念的な媒体として表現する．

– 特定の活動に関する記録は，そのプロセスを通過した情報の時間的な記録によって行う．

- 本論文で提案したプロセスと入出力対象に関する概念は，プロセス代数の CCS に基づいたものである．入出力における各プロセス動作及びシステム全体の活動を CCS に基づいて定式化した．この定式化を行うために，活動中の全てのプロセスの識別子の集合を持ち，プロセスの生成や消滅を実際に管理するプロセスマネージャの概念を導入した．プロセスマネージャの概念を用いることによって，プロセスから一つのプロセス，プロセスの集合，特定の種類のプロセス，などへの情報の入出力．プロセスの生成，終了，他のプロセスからの強制的なプロセスの停止，プロセスが活動しているかどうかの判定などの，プロセスの動作を表現する基本モデルを表現することができた．ここで表現されたプロセスマネージャの各機能は，基本モデルを実装する際の重要な課題となる．

- 本論文で提案したプロセスと入出力対象に関する概念は，オブジェクト指向的な考えでは，まず抽象的な入出力の対象のクラスを定義し，その下位型として本論文で提案したプロセスと入出力対象をモデル化する．具体的な設計生産活動モデルで用いられる作業プロセスや，CAD/CAM システムや設計活動者などの具体的な入出力対象は，これらのプロセスと入出力対象との下位型としてモデル化される．

- 本論文における基本モデルの特徴は，複数のプロセスが並列して動作し，それぞれ独立して情報の入出力行うことによって並列活動を表現することができること，並行動作するプロセス同士の時間的相互関係は，特定のプロセスが次のプロセスを起動する際の時間的な前後関係として表現できること，ルーリベースシステム以外の，多様な考え方のモデルの表現が行えること，設計生産に関与する様々な対象に対する広い表現能力を備えること，この概念に合わないモデルもブラックボックスである入出力の対象として表現することで，複数の概念に基づいたモデルも組み合わせることができる，などがある．

- 3.1.2でまとめた課題に対して他のモデルとの比較を行い，並列プログラミング言語と同様に本論文で提案した基本モデルが有効であることを明らかにした．

- 本論文で提案した基本モデルでは，情報以外の物の移動を直接表現することが難しいこと，単純に並行活動するプロセスを多数記述するだけでは，モデルの複雑化が懸念されること，安定性の数学的な解析が難しいこと，ルールベースシステムの表現が難しいこと，などの限界が明らかになった．

- 基本モデルでは完全に表現できないため，形式仕様記述言語の課題として時間的な情報を取り扱う必要があること，動作している同種のプロセスの識別が一意にできる仕様が必要であること，入出力される情報が，単なるシグナルレベルでなく，より高度な情報であることが必要である点が判明した．

4　設計生産プロセス記述言語

1.2では，設計生産の活動を表現して活用を計ることの問題点について述べた．また2.2において，設計生産における活動を表現する基本モデルに基づく形式仕様記述言語の問題点の分析を行った．本章ではそれらの問題点を考慮して設計生産活動に関する形式仕様記述言語の提案を行う．この言語を設計生産プロセス記述言語 (Engineering Process Description Language : 以下 EPDL と呼ぶ) と名付ける．

本章の構成は，4.1において，形式仕様記述言語を用いた既存の研究における分析を行い，解決すべき具体的な問題点を明らかにする．さらに4.1.4においては，設計生産以外の領域における形式仕様記述言語による表現の活用に関して述べる．次に 4.2 において，第3章で提案した特定の活動における情報の流れをプロセスとして扱う設計生産コミュニケーションモデルの基本モデルに基づいて設計生産プロセス記述言語を提案し，さらに 4.2.3 において，モデル化の例を上げて言語の性質について述べ，4.3 では提案した設計生産プロセス記述言語と従来からの研究とを比較して EPDL の特徴及び限界を明らかにする．

4.1　設計生産活動を表現する言語技術に関する分析

2.2でまとめたように，設計生産における活動を表現するためには，各種活動とその活動に関与する対象とを表現する基本モデルに基づいた言語を開発する必要がある．本節では，まず 4.1.1において既存の研究報告に関して考察を行う．その分析に基づいて 4.1.2において，本論文で解決すべき具体的な問題点を明らかにし，本論文における方針を定める．また 4.1.4においては，設計生産以外の領域における形式仕様記述言語を用いた研究に関して述べる．

4.1.1　設計生産活動を表現する形式仕様記述言語に関する問題点

本節では，設計生産活動の表現に形式仕様記述言語を用いた既存の研究に対する考察を行う [菊地 95a]．設計生産の活動表現に形式仕様記述言語を用いる理由として，

- 言語の仕様に従って表現を行うことでモデルの一貫性を維持する
- モデルの構造化を実現することでモデルとしての完成度を上げる．
- 意味表現における曖昧さを除くことで規格化を容易にする．
- 計算機による記述の取り扱いを可能にすることで，計算機によるモデル化支援や，その記述に基づいたシステムの開発を容易にさせる．

などが上げられる．

ここでは，言語仕様の背景として用いられる基本モデルではなく，言語の特徴，構造と機能との点から考察を行う．具体的には 2.2で述べた形式仕様記述言語を用いて表現するために必要な規則 [二木 92] の次の項目に関して比較を行う．

構文規則　どのような記号の並びが仕様文書であるかを定める構文規則．通常，形式文法 (生成文法) で与える．

意味モデルと意味規則　仕様の意味モデル

構造化方法 何を単位にして仕様をモデュール化し，それらのモデュールをどのよに組み合わせて仕様を構成するかを定めた規則.

考察の対象として，現状で言語仕様の詳細が得られている以下の研究 (言語仕様) に対して考察を行う.

- FDL [今村 92][今村 94]
- CIMOSA [Ver94]
- COSL [PK89]
- EXPRESS-P [FM94] [MFF92]

F D L

FDL [今村 92][今村 94] は，制約記述によってオブジェクト (製品) を記述する対象指向言語である．記述対象として設計データ及び設計モデルの宣言を行う.

FDL では部品オブジェクトを，そのオブジェクトに属する部品を列挙するパーツスロット (parts slot) と，その部品に関する情報を記述するパラメータスロット (parameter slot) によって記述する．パーツスロット (parts slot) に格納される部品はサブオブジェクトと呼ばれ，別途定義するかライブラリに登録されたものを利用する．補助手段として，ホーン節で記述されるメソッドの使用も可能である．さらに関係データベースの検索機能を備える.

FDL の構文は，設計モデル記述向けの書式を取っており宣言的に記述される．その構文仕様を [今村 92][今村 94] から図 16に引用する

FDL における構成単位は，部品オブジェクトである．このオブジェクト間の関係は，主に制約を定義される．即ち，FDL では部品の親子関係をパーツスロットによって，各部品間の関係を制約条件によって規定することで一つの設計オブジェクトを定義する．FDL で表現できる制約には次のものがある.

単一属性制約 属性の値について，型，単位，値の範囲などについての制約が宣言される.

同一性制約 2 つのスロットに格納されるオブジェクトが同一であることを宣言する.

多属性制約 データの整合性管理，仕様の表現，設計知識の表現を行う.

FDL では，記述されたモデルはコンパイルによって中間言語に変換され計算機の支援によって記述内容の一貫性のチェックを行うことができるが，多数のオブジェクトにまたがる多属性制約を用いて表現を行った場合に，その制約が意味している内容と，その制約が結果的にどういった影響を及ぼすかを仕様の作成者が容易に把握できるかどうかが問題点となる．このような問題に対しては，制約事項を個別にだけでなく何等かの形で構造化して認識しやすい仕様にする必要があると考えられる.

CIMOSA MSS 言語

CIMOSA[Ver94] で用いている MSS は，CIM システムの入出力に対する仕様の記述を目的として開発された．CIMOSA では，全ての製造システムは，重複しないドメイン (すなわち独立した機能領域) の集合として構造化される．各々のドメインは，並行ないしは連続的なドメインプロセスの

```
class <class name> has
   [super <super name> [, <super name>}; ]
   [view <view name> [, <view name>}; ]
   [slot <class slot> [, <classslot>}; ]
   {:<head> :- <body>;}
   {:<body>;}
instance
   [part <part definition> {,<part definition> };]
   [parameter <parameter definition>
                    {,<parameter definition> };]
   [database <item definition> {,<item definition> };
                        [<retrieval formula>;]]
   [where
        <slot definition>==<slot definition>
        {, <slot definition>==<slot definition>};]
   {<constant formulas>;}
   [method
      {:<head> :- <body>;}
      {:<body>;}]
   [local
      {:<head> :- <body>;}
      {:<body>;}]
end.

<item definition>::=<item name><item type><unit>
<item type>::=real¦integer¦string
<part definition>::=<part name>:=#<part class name>
<parameter definition>::=<parameter name><type>
                    [<parameter constraint>]
<type>::=real¦integer¦string¦list¦internal
<slot definition>::=<slot name>{!<slot name>}
<slot name>:=<parameter name>¦<part name>

[ ]は省略可能，{ }は0回以上の繰り返しが可能
```

図 16: FDL の構文定義 (一部)

有限な集合として構成される．ドメインプロセスは，部分的にはサブプロセス (すなわちビジネスプロセス)，かつ／またはアクティビティの順序集合である．アクティビティは仕事を実行する，即ち時間とリソースを必要とし，オブジェクトビューを使用，消費，操作，または生成する．オブジェクトビューは，企業のオブジェクトの素材化である．それは，素材 (物理的オブジェクト) もしくは情報 (オブジェクトの記述) のどちらかで有り得る．プロセスはイベント，すなわち要求又は指示または何かを実行することを要求するシグナルによって起動される．アクティブリソース，すなわち関数的な操作を実行するリソースは，機能エンティティと呼ばれる．最終的には，ドメインはイベント又はオブジェクトビューのみの (同期又は非同期の) 交換の形態で相互通信を行う．

CIMOSA で用いているモデルの基本構造は CIMOSA Cube と呼ばれ，次の３つの独立かつ相互に関係するモデリングレベル：

- requrirements definition
- design specification
- inplementation description

次の３つのモデル階層：

- generic buildling blocks
- partial models
- particular models

次の４つのモデリングビュー：

- function
- information
- resource
- organization

から構成される．

全ての製造システムは，ドメインと呼ばれる非重複領域，すなわち独立した機能的なエリアのセットに組み立てられることができる．

モデリングアプローチは，次の３つの基本モデルに基づいている．

Besuness Process 工業，生産の計画と制御，及び生産プロセスとして参照される．

Functional Entities 情報の受領，送信，操作，貯蔵を行う企業の動的なリソース

Eterprise Object 企業活動で用いられる任意の種類のオブジェクトの表現

CIMOSA における MSS 言語の意味モデルは，全体としてはドメインの集合としてモデル化される．ドメインは，大規模なモデルを別個に精察し，コンパイルできるように取り扱い，管理しやすいサイズへと分解する．ドメインプロセスとビジネスプロセスは，両方とも言語の共通のエンティティとして，プロセスとして表現される．

MSS 言語では，始めから多様な表現能力を目的としたため，ドメインとプロセスが複雑に関係するような場合に表現の複雑化が懸念される．また，リソースと情報の入力の区別を付けて表現能力の向上が行われているが，対象の構造変化までは表現しきれていない．

```
<Domain>::= Domain <D-identifier> <objectives>
        [<constraints>][<const-type-declaration>]
        Domain Processes : <process-list>
<const-type-declaration> ::= declaration section for constants,
        types and inported files as in ADA
<process-list> ::= <P-identifier> ¦ < P-identifier>, <process-list>
```

図 17: MSS 言語の構文定義 (ドメイン定義部分)

COSL

COSL [PK89] は，独立した製造アプリケーションにおける制御ロジックを *operation* の集合として定義する．COSL の *operation* の個々の実行は，制御オブジェクト全体を達成するために必要とされる制御ステップを表現する．

operation は，条件とアクションの集合を規定することによって定義される．各々の *operation* には，*operation* によって実行される制御ステップを記述する一意な整数番号と *operation* 名が与えられる．ある *operation* が可能ならば，すなわち *operation* の条件が真の場合，*operation* に関連するアクションが実行される．*operation* の後続する実行は，先行する *operation* の実行が終了した後に取って代わる．

COSL の仕様は，生産アプリケーションの制御のための *operation* を含むテキストファイルとして規定される．仕様の中では，その中で *operation* が定義される物理命令は，重要な意味は持たない．このことは，*operation* の間の優先する関係は，連続条件と *record operation* アクションを用いることを通じて明確に確立される．*operation* は，キーワードの順序集合として構文的に表現される．各々のキーワードは，キーワードに関連する様々な属性を認識するための指定された個数のフィールドが規定される．表 7に，COSL で用いられるキーワードとキーワード毎のフィールド記述を [PK89] から引用する．

COSL では，特定の条件によって起動される *operation* を羅列することによって，全体としての活動を表現する．*operation* 相互の関係は，*operation* によって起動されるイベントによって条件が満たされるかどうかの関係のみで，特定の *operation* から他の *operation* を明確に示すような関係は表現されていない．実際の実装では，この記述から C などの計算機言語のプログラムを生成することによって利用する．

本論文の目的である，設計生産における活動の表現を考えると，COSL では，予め決められたシグナルに摸した命令しか実行することができない．また独立したプロセスの概念が存在しないため，プロセス相互の関係が表現できていないなどの問題点が考えられる．

EXPRESS-P

EXPRESS-P [FM94][MFF92] は，プロセス記述を可能にするために製品モデル仕様記述言語である EXPRESS [ISO95b][菊地 93] のエンティティ記述部分を SDL [CCI88] などを元にして拡張したものである．

EXPRESS-P は，プロセスの基本的な振る舞いを定義するために，プロセス間の階層的な交信の

KEYWORD	field 1	field 2	field 3
op name (operation name)	operation description	-	-
op number (operation name)	integer (operation identifier)	-	-
sc (sequence condition)	sequence variable	L or G (mode)	-
pc (physical condition)	sensor name	0 or 1 (sensoe value)	T of F (mode)
cc (counter condition)	counter variable	integer (count value)	T of F (mode)
pabd (physical action befor delay)	actuator name	0 or 1 (actuator value)	-
paad (physical action after delay)	actuator name	0 or 1 (actuator value)	-
delay	integer (real-time delay)	-	-
clear counter	counter variable	-	-
ro (record operation)	sequence variable	-	-
or (separator of multiple condition sets)		-	-
stop control	termination state	-	-
counter	counter variable	-	-

表 7: COSL におけるキーワードとフィールド記述

構造，すなわちデータの発信と受信のための関係を提供する．これによって静的なエンティティは，連続的な交信プロセスを表現する動的なエンティティとなる．この文脈では，動的なエンティティとは *system* 又は基本的な *process* である．システムとプロセスは，他のプロセスと交信するための *interface* を持つ．*system* は交信の構造を定義する．システム定義の中で，エンティティは非同期のエンティティ間の交信のために交信チャンネルによって接続される．チャンネルは，システムエンティティのための関係を持つエンティティの間でだけ定義される．*process* は，他のプロセスへ／からインターフェースを通じてメッセージを送ったり受信したりする連続的な実行文による振る舞いを定義する．

図 18に，基本的な拡張部分と，インターフェース定義とチャンネル定義の例とを示す．全ての動的エンティティ(システムとプロセス) は，交信インターフェースの集合を持つ．インターフェース記述は，インターフェースの識別子と，可能であればこのインターフェースを通じて受信することができるシグナルの集合によって定義される．各々のシグナルに対して，対応するデータ型が定義される．インターフェースが常に属性によって修飾されるため，異なったエンティティのエンティティ

```
ENTITY ...;
        ...
        <Interface Declaration>
        <Local Function Procedure>
        <System- or Process Declaration>
END_ENTITY;

                EXPRESS の拡張部分

INTERFACES
        interface1 WITH s1, s2(INTEGER);
        interface2 WITH s1;
END_INTERFACES;

                INTERFACE の定義部分

entity1 : ETYPE1;
others : set [0:?] OF ETYPE2;
....
CHANNELS
        entity1.interface1 <-> others.interfac2;
END_CHANNELS;

                CHANNELの定義部分
```

図 18: EXPRESS-P における言語拡張部分，インターフェース定義，チャンネル定義

タイプインターフェースへの参照では，同じインターフェースの識別子を用いることができる．

プロセスはエンティティごとに定義され，キーワード PROCESS 及び END_PROCESS に囲まれた中で連続的な活動を記述する．プロセスの実行部の記述は，基本的には EXPRESS における手続きの記述と同じであり，プロセスの同期とタイミング処理のための拡張がなされている．プロセスは，対応するエンティティがインスタンス化された時に活動化され，最終実行文を実行した後で消滅する．このプロセスの活動を停止するための実行文が拡張されている．プロセスからのシグナルの発信は OUTPUT interface1.s1(v1); のような出力命令文で実行される．この例では，interface1 で参照されているインターフェースに対して，引き数 v1 を持ったシグナル s1 が発信されている．プロセスの受信は入力命令によって実行される．

プロセスの表現は各エンティティ毎に行われる．インターフェース定義，局所関数及び手続き，システム，プロセスは，EXPRESS における一般的なルールに基づいて，継承され再定義されることができる．

また既存の EXPRESS による記述と，プロセスに関する記述を別々に行っておいて後から簡単に統合するために，予め定義されているエンティティスキーマを参照して，プロセスのみを定義するスキーマを用意することが可能になるような構文上の拡張がなされている．

EXPRESS-P は，基本的には型の定義しか行っていない．即ち定数や変数が存在しないため，実際の通信構造を表現するためには物理ファイル上で表現しなければならなかった．EXPRESS-P では，エンティティ同士の通信を行う構造の極めて堅固な型枠を初めに定義し，実際の構造は，物理ファイルなどの上で定義される．この際の問題として，エンティティから見た他のエンティティの共有や参照などのエンティティ同士の関係と信号を発信又は受信するための交信の構造が 2 重に定義されることが考えられる．EXPRESS の物理ファイルにおける参照構造の実現は，例えば SELECT ステートメントの解釈に関して不安定な部分が見受けられるため，通信構造の定義に関しても，同様の問題を持つ可能性がある．しかし，この問題に関する部分の記述は全く行われていない．

次に STEP 物理ファイルは，特定の時間における製品モデルデータのスナップショットを示すものであるので，そのデータ構造が時間の進展とともに動的にどのように変化するかは明確には表現できない．また入力を受け付けるためのステートメントを用いて受信を陽に記述しなくてはならないため，入出力のタイミングの記述が難しいと考えられる．

4.1.2 本論文における設計生産活動に関する言語技術への課題

前節 4.1.1で行った既存の各言語仕様に対する考察から，設計生産活動の表現を行う言語として解決する必要がある課題について整理する．問題となったのは，主に言語要素，複雑化の懸念などであった．課題全体をこれらの項目に分けてから，さらに細目に関してまとめる．

1. 言語仕様全体としての課題

   - 多様な表現能力を目的として表現できる要素の種類を始めから多くすると，モデルの規模が大きくなった場合の表現の複雑化が，また，言語仕様間での相互の関係が複雑化してしまうことも懸念される．言語の仕様としては，単純で少ない数の言語要素による構成を目指す必要がある．
   - 多数のオブジェクトにまたがって属性間の制約を用いる表現を行った場合に，その制約が意味している内容と，結果的にどういった影響を及ぼすかを仕様の作成者が容易に把握できるかどうかが問題点となる．このような問題に対しては，記述表現を個別に行うだけでなく，構造化された認識しやすい仕様にする必要がある．
   - EXPRESS-P における物理ファイルにおける参照構造の実現は，例えば SELECT ステートメントの解釈に関して不安定な部分が見受けられるため，通信構造の定義に関しても同様の問題を持つ可能性がある．本論文では，全体としてのモデルの参照構造と，情報や物の流れの構造をと二重に定義する方法は用いない．

2. プロセスの表現

   - プロセスを独立した存在として表現出来ない場合，そのプロセス相互の関係が表現できない問題が考えられる．従って，活動中のプロセスを陽に表現できるようにする必要がある．

3. 設計生産活動に関与する対象の表現

   - これまで考察を行った形式仕様記述言語では，基本的には静的な対象間での活動を対象としており，活動の主体の動的な構造変化までは表現しきれていない．設計グループの活動を対象と考えた場合は，この設計グループの構造が，設計活動の進展に従って，動的に組み替えられる状況が考えられ，このような活動を表現する必要があると考えられる．

- 入力を受け付けるためのステートメントを用いて受信を陽に記述する方式では，イベントループを仕様作成者が記述しなくてはならず，タイミングの取り方の記述が難しくなることが考えられる．このため，設計生産活動などの表現では，情報の受信を受信命令を用いない方式が必要となる．

4. 入出力される情報や搬送状況の表現

- 本論文の目的である設計生産における活動を表現するためには，プロセスや活動に関与する対象の間で，予め決められた通信回線を通る信号に摸した命令しか送れない方式では，表現能力が不足すると考えられる．従って，送受される命令は，言語の仕様として特定の種類を予め決めておくのではなく，ユーザが規定できるようにする必要がある．
- 実際の生産システムにおける加工プロセスでは，被加工物が加工システムに供給されたり消費される．これらのリソースの入出力や搬送の状況も表現できるようにする必要がある．

5. 物理的な表現や実装への配慮

- EXPRESS-P における物理ファイル上での通信構造の表現のように，特定の時間における通信構造を特定のフォーマットに示す方式を用いたとしても，その構造が時間の進展とともに，動的にどのように変化するかは明確には表現できない．特定時間の構造の表現は必要であるが，その構造は動的に変化できる必要がある．
- EXPRESS-P では，エンティティ同士の通信を行う相手のエンティティの種類を初めに定義し，実際の構造は，物理ファイルなどの上で定義される．この時の問題として，エンティティから見た他のエンティティの共有や参照などのエンティティ同士の関係とシグナルを発信又は受信するための交信の構造が 2 重に定義されることになる．このような相互参照の複雑化は避ける必要があると考えられる．

設計生産の活動を表現するためには，上で挙げた問題点を克服できる言語仕様が必要になる．本論文では 4.2章において，ここで挙げた課題を解決を目指して，プロセスに関する基本モデルを表現する言語仕様の提案を行う．

4.1.3 CAD データ／製品モデルデータ交換の現状

設計生産活動の情報表現に対して，まず CAD データの，さらに製品モデルデータの現状に対して考察する．

CAD データの交換では，相互のシステム間でデータの交換を行う際の中間的なフォーマットとして IGES[RHC90] が ANSI 規格として制定された．しかし IGES は，

- 形状データを固定長のレコードフォーマットでどのように表現するか，に関してしか規定していない．
- IGES データファイル内で用いられる形状要素の意味的な定義が行われていない．
- どのような値のデータが正常な IGES データであるかの規定がなされていない．例としては，円周上の３つの点の座標で円が定義される場合に，各点が一直線上にあってはならないが，このような状況を禁止するような規定が存在しない．

- どのような要素の組み合わせが，正常なIGESデータとして扱われるかが規定されていない．例えば，製図の中の特定の図形を，線分の集合として表現することができる．また，同じ図形を単なる線分の羅列として表現することができる．どちらの表現が推奨されるかの規定がなされていない．

等の致命的な欠点があった．このため，

- IGES準拠をうたうデータ交換プログラムが，自身の内部データとの入出力にしか注視しなかったため，実際には特定のIGESデータしか入出力できない仕様の入出力プログラムが開発されてしまった．

- IGESが持っている形状要素にCADシステムの内部データが含まれていない場合，逆にIGESの形状要素をCADシステムの内部データとして持っていない場合，何らかの形で近似操作が行われることになる．この近似操作の方法に関する規定が全くないため，一度IGESデータとして出力したデータを元のCADシステムに読み込みなおした際に，最悪の場合に元のデータが再現できない事態が起こる．

- 上記の問題から，IGESデータとして出力されたものであっても，相互にデータの交換が行えないことが頻発したため，個々のシステムが出力するIGESデータフォーマットを解析してデータ要素の不一致を解消させてお互いのデータを交換するIGES-to-IGESデータプロセッサまでもが開発されている．

などの問題が起こった．

IGESの方法論に対する反省からSTEPの開発が進められることになった． STEPは，製品データを交換するための規格であるとの誤解を一般に持たれやすいがその本質は製品データのフォーマットではなく製品データが持つ意味の規格化にある．

STEPでは，IGESでの最大の問題点であった，モデルの個々の要素及びモデル全体として意味とそれぞれの要素のデータが正常な状態とを規定するために，製品モデルの意味的な情報の表現が形式仕様記述言語EXPRESSによって第一に行われ，物理的な実際のデータフォーマットは，そのデータモデルの外部表現として写像の形で定義されている[ISO95a][ISO95b][ISO95c][鈴木 93]．

図19に[木村 95]及び[Fow95]からSTEPの全体構造と各パートとの関係を示す．STEPにおける製品モデルは，まず各アプリケーションの分野で共通に用いるモデルの基本的な構造，幾何形状，製品構造などの資源モデルの要素を予め定義した上で，実際の分野では，それらの資源モデルを，一部を変更して組み合わせることによって製図，３次元の形状表現，FEM，板金，自動車全体の製品データなどの種々のアプリケーションの領域で用いられるモデルとして開発する手法を用いている．このことによって，STEPデータモデルに基づくアプリケーションシステムは，データフォーマットの形式が異なっていても，本質的には同じ意味を持ったモデルを扱っていることになる．従って，製品モデルに関しては，予め意味的な内容の基準を持った上で，その意味によって定義されたデータを交換することが可能になる．

ここまで述べた製品モデルに関する動向をまとめると，次の項目となる．

- ある基準に基づいて特定のモデルの意味的な質を保証している．

- 製品に関する知識の蓄積／共有を行うための記述表現の技術が確立された．

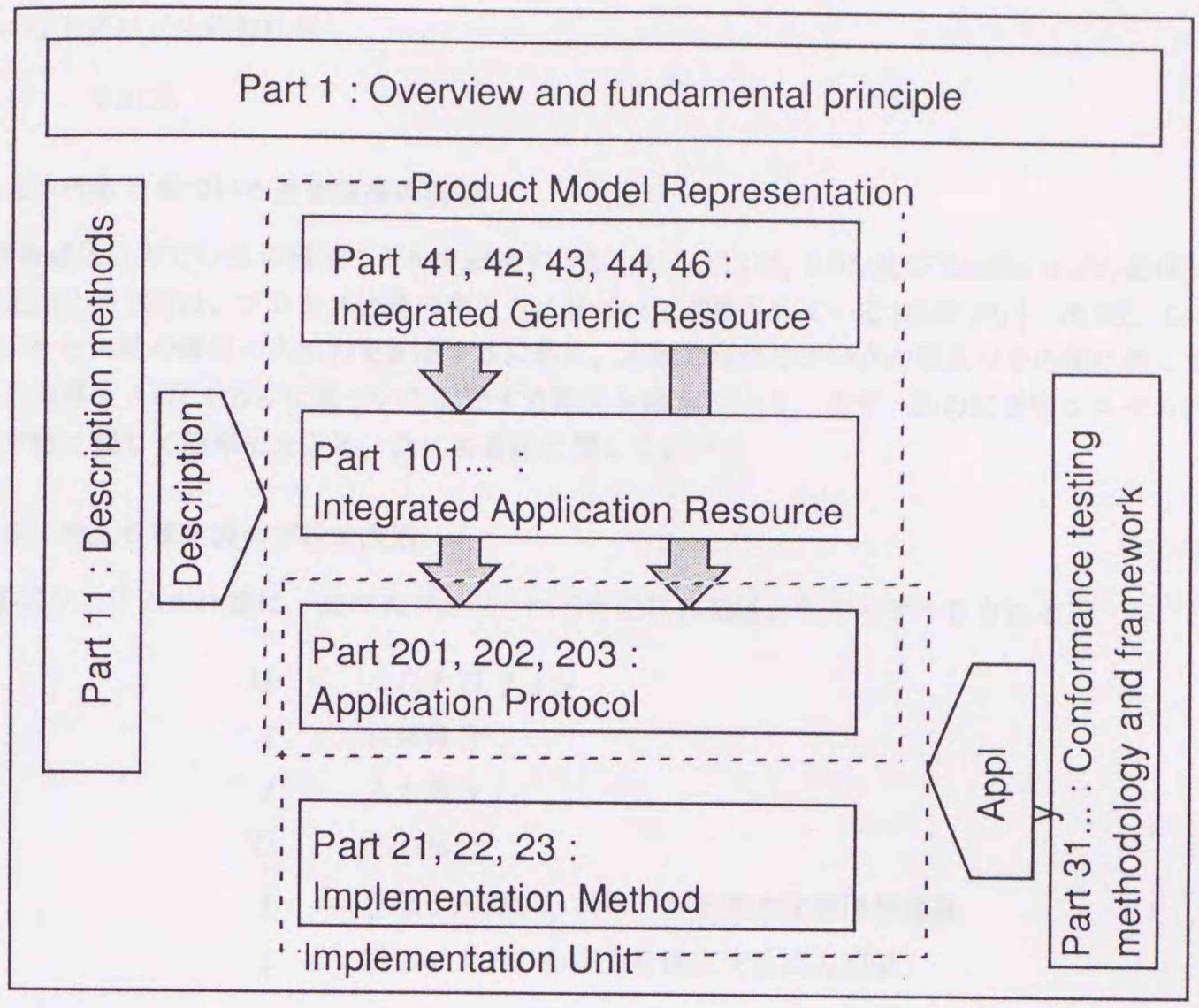


図 19: STEP 規格の構造

- ある製品モデルデータが正常かつ妥当であるかを判定するための，基準と判定方法が確立された．
- 製品モデルに関する意味的な情報を計算機処理することが可能になった．
- 共通の製品モデルに基づいたアプリケーション開発を推進するための基盤が確立された．

4.1.4 設計生産以外の領域における形式仕様記述言語の活用

仕様の記述に形式仕様記述言語を用いる研究として，設計生産の活動表現の目的以外では幾つかの報告がなされている．本節では，参考として次の目的の分野での言語に関する概要について述べる[菊地 96]．

- プロセス代数に基づいた通信規格の表現
  - LOTOS
  - Estell
  - SDL

- 電子デバイス設計仕様
  - VHDL

プロセス代数に基づいた通信規格の表現

情報通信分野でOSIの階層モデルを記述するためのLOTOS，SDL及びEstelleなどの通信プロトコル記述仕様言語は，プロセス代数の考え方に基づいて定義されている[水野90] [二木90]．LOTOSは，プロセス間の情報の入出力を記述する以外に，入出力されるデータの型及びその型に対して適用される演算をACT ONEに基づいて記述する能力を備えている．まず，初めに通信システム仕様定義の方法に関して簡単にまとめ，次に各言語に関して述べる．

通信システム仕様定義の方法の大別

通信システムの仕様は，次の式で表現される有限状態機械としてモデル化される．

$$
\begin{aligned}
M &= < S, I, O, f, g > \\
S &: \text{状態集合} \\
I &: \text{入力集合} \\
O &: \text{出力集合} \\
f &: \text{状態と入力から状態遷移を興す状態遷移関数} \\
g &: \text{状態と入力から出力を決定する出力関数}
\end{aligned}
$$

さらに挙動を表現する方法に応じて，次の2種類の方法でシステムを規定する．

外部挙動規定法 実装を特に意識せずに，通信を行う者同士が互いにやり取りする情報の順序関係だけを規定する方法．上記の例では，状態Sを陽に記述せず，入力Iと出力Oの組み合わせにより有限状態機械を定義する方法に該当する．LOTOSがこの方式を用いている．

遷移機械規定法 実装をある程度意識し，動作を抽象的な状態遷移機械として規定する方法．上記の例では，状態Sを陽に記述し，状態Sと入力Iと出力Oの関係によって有限状態機械を定義するものである．SDL及びEstelleがこの方法を用いている．

LOTOS

LOTOS [ISO89] は，FDT (Formal Description Technique) の一種としてISOによって1981年から1988年にかけて開発され，現在国際規格として勧告されている．またグラフィック構文を有するG-LOTOSも研究されている．その目的は，OSIアーキテクチャにおける各種のサービス定義やプロトコル仕様を形式仕様記述言語を用いて記述することにあるが，一般的には分散型の情報処理システムへの適用が可能である[高橋90a]．

LOTOSでは，システムの記述をプロセスの観点から行い，外部から観測される振る舞いとしての通信能力を記述することでシステム仕様の記述を行う．このプロセスの振る舞いは，MilnerによるCCS (Calucuas of Communicating Systems)[Mil89] に基づいており，記述された仕様の解析に有効な理論的枠組みを提供する．LOTOSにおけるプロセスとは，その環境 (対象としているプロセスの外側全体) における他のプロセスと通信を行う抽象的な実体を指す．プロセス間の通信は，「ゲー

ト」と呼ばれる作用点で起こり，通常はそこでデータの交換が行われる．インタラクション (通信による相互作用) の基本単位は,「アクション」(あるいは「イベント」) と呼ばれ，ゲートとデータ値の組みから成る．アクションは，原子的で同期的なインタラクションである．アクションが原子的であるとは，

1. 瞬間発生である．
2. 細分化できない．
3. 他のアクションと時間的な重なりがない．

ことを意味する [高橋 90a].

LOTOS は，外部挙動規定法による記述を行う．すなわち，そのシステムとやりとりする外部から観測可能なイベント間の時間順序を規定することによってシステムの仕様化可能であるとの視点からモデル化を行う．LOTOS では，通信システムの記述を時間順序の概念のもとで外部との観察点でのイベントの発生順序に従って行う．プロセス (通信プログラム) に対して環境と通信するブラックボックスとみなすことができる．プロセスは，インタラクションの元になるイベントの列によって規定し，データに関しては SDL と同様に抽象データ型を採用している．

LOTOS における値，値式，データ構造の表現は，抽象データ型 (ADT : Abstruct Data Type) 記述言語 ACT ONE に由来する．ADT は，互いに関連したデータを結合し，それらに関して適当される演算の構文と意味を定義することを目的とする．ACT ONE は，パラメタ型仕様及び非パラメタ型仕様を書くための代数的仕様記述法であり，構造的な仕様を生成するために次のような特徴を持つ．

- ライブラリとして予め定義されているデータ型の使用
- 仕様の結合
- 仕様の名前の付け替え
- 仕様のパラメタ化
- パラメタ型仕様の具体化 (actualization)
- 演算とソートによる仕様の拡張

LOTOS における基本的な動作表現では，

$B^1, B^2, \ldots, B^n$ ： 動作表現

$g_1, g_2, \ldots, g_n$ ： ゲート名

の記法に基づいて．次のような CCS を拡張した記法が用いられている．

1. $a; B : actionprefix$
   イベント a が生起した後の動作は Bによって規定される．
2. $B_1[\,]B_2 : choice$
   B_1か B_2のどちらか一方の動作表現でイベントが生起できる時，生起できた方の表現が選択され，それ以降の動作がその選択された方の表現で規定される．

3. $B_1||[g_1, g_2, \ldots, g_n]||B_2$: $parallel\ over g_1, g_2, \ldots, g_n$
   B_1か B_2のどちらかで生起できるイベントの内，ゲート $g_1, g_2, \ldots, g_n$で生起するイベントは同期して生起しなくてはならない．それ以外のイベントは，*interleave* して生起する．これは，B_1と B_2がゲート $g_1, g_2, \ldots, g_n$を巻き込むアクション (基本 LOTOS の場合，これらは $g_1, g_2, \ldots, g_n$ そのもので記述される純粋な同期アクション) に関して同期しなくてはならないような並列合成を表す．

4. $B_1|||B_2$: *interleave*
   B_1か B_2のどちらかで生起できるイベントがあるとき，それらが interleave して生起する．これは，なんら互いに同期関係の無いプロセスが動作している状況を示す．

5. $B_1||B_2$: *syncronize*
   B_1か B_2の両方で同時に生起できるイベントしか生起しない．これは，相互に依存する同期関係の無いプロセスが動作している状況を示す．

6. $hide[g_1, g_2, \ldots, g_n] in B$: *hiding*
   Bで観測されるイベントのうち，ゲート $g_1, g_2, \ldots, g_n$で生起するものを外界から遮蔽する．

また LOTOS におけるプロセスの一般形は，次の図 20の書式で表現される．

```
process P[ a, b, c] (...) ... : =
        <behavior-description>
endproc
```

図 20: LOTOS におけるプロセスの一般形

これに対応して，キーワード”specification”及び”end-spec”があり，これらはトップレベルのプロセス (つまり仕様自身) の定義の始まりと終わりを示す．かぎ括弧の中は仮ゲート数，カッコの中は仮変数引数である．

S D L

SDL [CCI88] [若原 90] は，CCITT において交換機の仕様を記述するために開発された，遷移機械規定法による記述言語である．SDL は，有限状態機械に加え，変数，動作，及び述語の概念を加えた拡張有限状態遷移機械のもとに，フローチャートに似たグラフ表現と同等の意味を持つテキスト表現で，通信システムを記述する．

SDL の言語仕様はシステム，ブロック，プロセス，及び手続きの４つの段階に分けて規定されている．またデータ自身に関しては抽象データ型を利用し，これは LOTOS におけるものと同等である．

記述の構成と方法を 2 つの観点から述べる．

静的構造記述 システムを構成する機能単位であるブロックやプロセスの間の静的な信号送受関係によりシステムの枠組みと構造を表現する静的構造記述

最下位レベルのブロックは必ず一個以上のプロセスを持つ．記述対象としない部分はまとめて環境 (ENV) と呼び，環境との間にチャネルやシグナルルートを設定することもできる．

システム動作開始時に存在するプロセスインスタンス数と，システム動作中に存在するプロセスインスタンス数の上限値とを定義することができる．

動的構造記述 各プロセスの動的な動きを表す動作記述プロセス動作部分は，型を用いてその動作のひな型を定義するが，各型においては状態ごとに信号入力に対する動作処理を手続き的に記述する．各型によって定義された動作をとるプロセスの実体はインスタンスとして必要な数だけ生成することができる．受信入力を陽に記述していない信号を受信したときには，その信号を無視して棄却してもとに戻る．

SDL にはタイミング処理などの記述のためのタイマ機能がある．タイマは SET 処理によって起動され SET 時に指定された時間を，その後経過するとタイマ信号を発生してそのプロセスに出力し動作を停止する．タイマ信号を発生するまえに RESET 処理が実行されるとタイマは，その動作を停止する．

プロセスの記述の複雑化を回避するための手段として手続き (PROCEDURE) とマクロ (MACRO) がある．SDL/GR では分割された各部分を連結させるためのコネクタを使用する．コネクタには，その識別のための名前と連結先を含むページ番号を付ける．

Estelle

Estelle は，ISO において OSI のプロトコルを規定するために開発された．遷移機械規定法によるものである．Estelle は，ISO ですでに標準化が済んでいる Pascal の言語機能に対して，状態遷移機械の概念などの通信システムの記述能力を追加して拡張している．Pascal に追加された状態遷移を記述するための構文としては，現在の状態 (from 節)，入力インタラクションプリミティブ (when 節)，発火述語 (provided 節)，遅延条件 (delay 節)，優先条件 (priority 節)，次の状態 (to 節) などがある．

電子デバイス設計仕様定義言語 VHDL

VHDL(VHSIC Hardware Description Language) は，合衆国政府によって行われた VHSIC(Very High Speed Integrated Circuit) プログラムの成果として生まれた．その目的は，以前よりも短期間で，より高度なマイクロエレクトロニクス部品をオペレーショナルシステムに組み込む能力を開発し，企業内，企業間でコミュニケーションの改良によって開発過程の合理化を計ることである．VHDL の開発は，VHSIC のスポンサードによって，1981 年の 6 月マサチューセッツのウッズホールで行われた研究会で正式に始められた．この研究会の目標は，業界の標準となり技術的に独立しマイクロエレクトロニクス業界の要求に答えるハードウェア記述言語の開発である．

米国空軍は，このウッズホール研究会の要求書を元に，マイクロエレクトロニクス業界の検討を経た後に 1983 年に提案要求書を完成させた．同年 7 月，インターメトリック社，IBM 社，テキサスインスツルメント社で構成された開発チームが新言語の開発とサポートソフトエアの開発を委託された．言語仕様は 1985 年 8 月に最終バージョン，VHDL バージョン 7.2 がリリースされた．1986 年 5 月 IEEE は VHDL の標準化作業を開始し見直しを計った．その後 1987 年 12 月に IEEE1076 規格として正式に承認されている [LSU89]．従って VHDL には大きく分けるとバージョン 7.2 と IEEE-1076 の 2 つのバージョンがあることになる [Jame89]．

[Car90] では，VHDL に限らない一般的な HDL(Hardware Description Language) を用いる利点として，次の点が上げられている．

1. 設計者の生産性の向上：設計者は HDL を使用することによりゲートレベルの記述を手作業で入力する必要がなくなり，設計期間の短縮を計ることができる．

2. 抽象度の高いレベルでの設計：RT レベル (レジスタレベル) に移れることで，設計者はゲートレベルで行っていたときよりも設計の構想のレベルですべき思考に，より多くの時間を使うことができる．

3. 設計品質の向上：自動最適化と，それが持つ様々な回路パターンを容易に試みることができる機能により，設計者は最終結果として，より高品質の回路を得ることができる．

4. 抽象レベル間の一貫性：RT レベルとゲートレベル間の自動的な橋渡しを行う機能により，論理合成前と後のネットワークモデルが等価であることを保障することができる．

5. シリコンに関する知識の必要性の軽減：設計テクノロジに依存する部分の作業は自動化されるので，設計者は各 ASIC ベンダのライブラリの特色にこれまでのように慣れる必要がなくなる．

6. 設計の再利用の促進：RT レベル記述はテクノロジに依存せず，本来はネットリストを読むよりも容易なため，設計モデュールの再使用がはるかに実際的になる．

7. テクノロジからの独立：テクノロジに依存しない方法でプロセッサの設計を行うことができる．従って現実に存在するシリコンテクノロジの，どの技術を使用するかの判定を，設計プロセスのより後方で決定することができる．また，テクノロジの差し替えを行うことが可能となる．クロック速度や遅延時間をテクノロジに依存しないモデルに取り入れることは，その時点でモデルのテクノロジ依存が生じることになる．たとえそれが，設計の目標や制約であろうとも，そのモデルをある時代のテクノロジに依存させてしまう．性能目標は論理合成の品質に関して重要な意味を持つが，それはモデル内部に持たずに，特定のプロセスに対するライブラリを目標とした枠内 (VHDL パッケージコンスタントや別の制約ファイルなど) でよりよく表現することができる．

特に VHDL を用いる利点としては，次のような項目が上げられている．

1. 標準／公開言語：
   標準言語のため，設計環境を変更しても設計を移し換えることができる．

2. 単一言語によるシミュレーションと論理合成：
   全ての設計活動に対して単一の言語のみしか修得する必要がない．

3. 米国国防省承認：
   MIL 規格 454 では，政府のプロジェクトとして設計される全ての ASIC は，VHDL でドキュメントを作成するように定めている．

4. モデルの利用性の高さとポータビリティ：
   VHDL のモデルの種々のソースは，新しい設計に対して多くの利益をもたらす．標準モデルの使用により，ベンダ，契約者，ツール間でのモデルのやり取りが行える．

VHDL におけるデザインエンティティ

VHDL においては，論理回路はデザインエンティティとして表される．このデザインエンティティは，マイクロプロセッサのレベルから AND ゲートのような単純な回路までを対象とする．デザインエンティティの記述は，入出力ポートに関するインターフェース記述と，内部のアーキテクチャ本体に関する部分に分けて定義している．

属性 アトリビュートはエンティティを表現するための値である．多くの属性が言語リファレンスマニュアル中で定義されている．またユーザ定義属性を使用することもできる．

アーキテクチャ宣言 アーキテクチャ宣言では，始めにチップの動作をアルゴリズムを用いて記述する．しかしこのレベルでは，実現されるロジックや動作の遅延がどうなるかは求めることができない．設計者はその後で，実際のゲートの組み合わせによる完全なアーキテクチャに置き換えていく．

インターフェース記述 エンティティ宣言では基本的にデザインエンティティの入力と出力のみを定義する．その値としては，ビット，ビットベクター，整数として定義できる．

複数アーキテクチャの定義 同一の入出力特性を持っているデバイスにおいて，異なる設計方針とアルゴリズムを適用する複数のアーキテクチャ定義を行うことができる．各特性を記述するアトリビュートを定義して，これに条件式を付加することで，特性記述を行う．

ポートマッピング 特定のエンティティの内部をさらに複数のエンティティに分解する場合，上位の信号が，下位のエンティティのどの信号にマップされるか指定することができる．

ブロック構造 VHDL では，言語の基本要素の境界と，アルゴリズムの組み合わせのためにブロック構造を用いている．各ブロックには，始めにラベルと宣言部分，その後に実行部分が存在する．エンティティ宣言によるインターフェース記述や，アーキテクチャ記述も一つのブロックとして扱われる．

プロセスブロック アーキテクチャの動作を記述するブロック．従ってアーキテクチャ宣言ブロックの中には必ずプロセス記述ブロックが存在する．

ま と め

通信システム仕様定義のための言語は，プロセス代数という明確な数学的な基盤を元にし，プロセス代数によって行われる演算を言語的に表現する方法を用いている．また，データ型の定義では ASN.1 と呼ばれる抽象型定義法を用いて (複雑ではあるが)，安定した定義を行えるようにしている．

しかし EXPRESS における型の定義は，数学的な背景を持たずに複雑な構造を言語で表現する方法を用いており，この部分に対する不安定さが指摘されている．本論文においても，数学的なあるいは理論的な基盤を元にしてその中での演算または活動などを直接的に言語によって表現する方法を取る必要があると考えられる．

電子デバイス設計仕様定義や通信システム仕様定義のための言語は，その通信構造が一定で変化しないことを念頭にしている．しかし，本論文では組織の構造が，活動の進展に応じて変化していく状況などへも対応する必要があると考えられる．

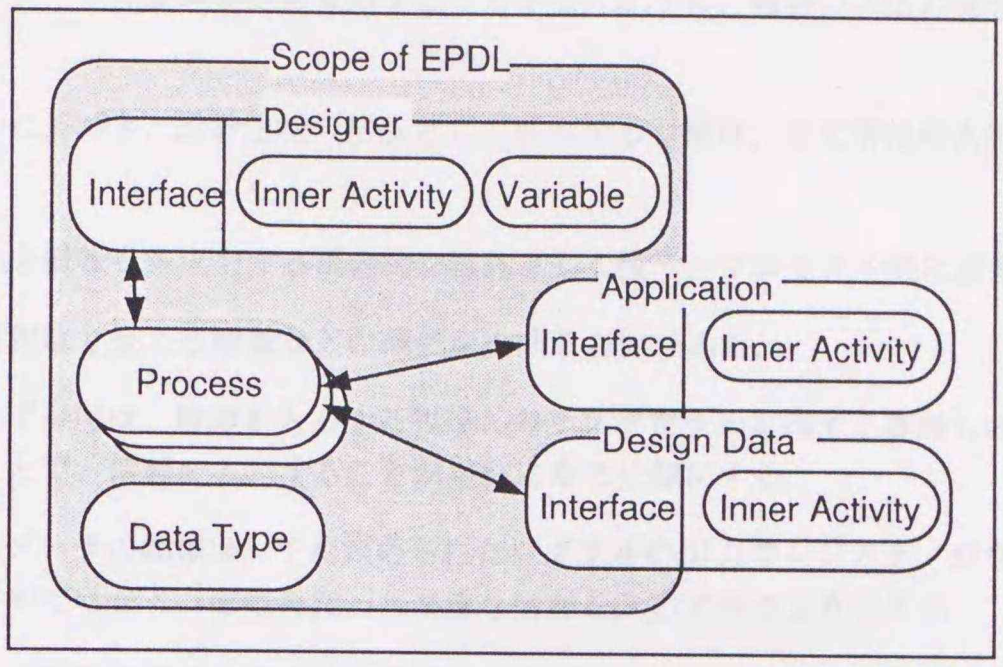


図 21: EPDL によって表現される対象と表現可能な領域

4.2 本論文で提案する設計生産プロセス記述言語

本節では，4.1.2 で得られた課題を克服するために，設計生産活動を表現するため第 3章で提案した基本モデルに基づいた設計生産プロセス記述言語の基本構造の提案を行う．

4.2.1 言語の概要及び基本構造

基本構造

3.2.1で提案した基本モデルに基づいて，設計生産における入出力対象及び情報入出力過程の表現を行う設計生産プロセス記述言語を提案する．以下では EPDL(Engineering Process Description Language) と呼ぶ．この言語は，1.3.1で述べたモデルの表現及び計算機を用いたシミュレーションを目的として開発され，次の 3 項目の記述を行う．

- 情報を入出力するプロセスの挙動
- プロセスから見た入出力対象の仕様
- 入出力される情報の型

初めに言語が表現する範囲を図 21に示す．第 3章で提案した基本モデルでは，外部に存在する入出力対象の内部活動は表現の対象としないこととしているが，言語仕様ではオプション扱いで，この部分の表現を必要に応じて行うことも可能としている．構文定義は，WSN (Wirth Syntax Notation) [ISO95b] を用いる．言語仕様の詳細については付録として Aにまとめる．

言語仕様決定の方針

言語仕様の決定では，前節での課題と 3.4.3 で上げた項目に従って次の点に注目して行った．

- 表現するための言語要素 (キーワードや表現用のブロックの種類) は，できるだけ少なくする．

- 通信構造やデータの参照構造が交錯することがないように，情報の入出力構造の表現だけに注目する．
- 基本モデルに基づき，設計生産の活動として関与する対象は，全て情報の入出力対象として表現する．
- 特定の活動を情報を入出力する概念的な媒体として扱う，プロセスを陽に表現する．
- 時間的な情報取り扱える関数などの機能を設ける必要がある．
- 本論文のモデルでは，情報を入力する相手とのリンクを予め定義する事無しに相手の識別子を指定することで，情報を入力することが可能になる仕様にする．
- 入力されるデータの種類は，予め定められたシグナルの出力やレジスタ／カウンタ値の変更などの低レベルではなく，属性を持った高度な情報を入力できるようにする．
- LOTOS は，CCS の概念を直接的に言語表現することを目的としているため，対象同士の接続状況を明確に規定して，各対象におけるアクションの相補的な関係を表現しなくてはならない．本論文では，対象を具体的に接続された通信機器ではなく，作業者や作業者のグループなどとしているので，表現も直接通信を行う機器同士の接続の形態に囚われないで表現する必要がある．従って，LOTOS 的な接続の表現を行う必要はないと考えられる．

言語で表現される範囲

本論文で用いている設計生産プロセス記述言語は，基本的には 3.1.3で提案した設計生産コミュニケーションモデルに基づくものであるが，3.2.2で提案したプロセスマネージャの挙動は，言語の仕様として即に実現されている状況として陽には表現されない．すなわち，プロセスマネージャによって実現される複数プロセスへの同時並行な情報の伝達や，プロセスの生成／消去などでの，特定プロセスからプロセスマネージャへのメッセージの発信の形での記述表現は行わずに，直接特定のプロセスから他のプロセスへと発信される形で表現される．また，プロセスの開始や終了を行う際にも，プロセスマネージャに対して終了を要求する形ではなく，開始や終了の相手を特定しない形の構文で表現する．

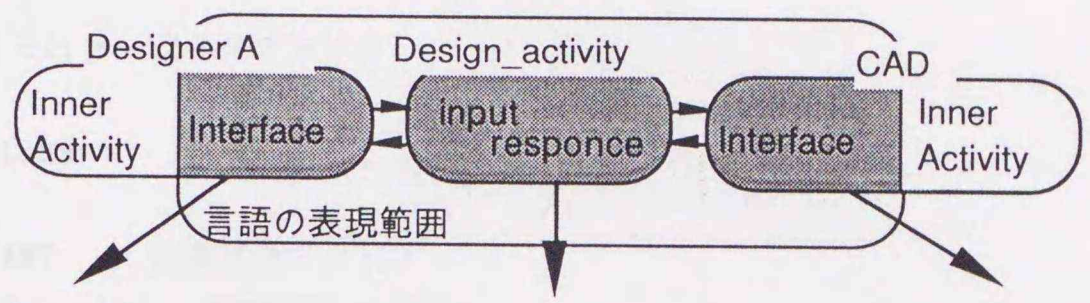
図 22に設計生産プロセス記述言語と基本モデルの対応関係を示す．

このことによって，設計生産プロセス記述言語を用いた活動表現では，プロセスマネージャに関する記述は，全く表現されないことになる．ただし，形式仕様記述言語を用いて表現された活動を計算機システム内で実装する場合には，逆にプロセスマネージャの実装方法事態が重要な課題になると考えられる．

言語のブロック構造

記述されるモデルは，ブロックの集合として扱われる．定義ブロックの種類としては，以下のものがある．

- タイプブロック
- クラスブロック
- 定数宣言ブロック




```
INTERFACE DESIGNER
MESSAGE
  input(argument:data_type);
END_MESSAGE;
END_INTERFACE;
```

```
PROCESS Design_activity
MESSAGE
  input(argument:data_type);
  responce(argument:data_type);
END_MESSAGE;
END_PROCESS;

Design_activity
    :input(argument:data_type)
BEGIN
END;
Design_activity
    :responce(argument:data_type)
BEGIN
END;
```

```
INTERFACE CAD
MESSAGE
 input(argument:data_type);
END_MESSAGE;
END_INTERFACE;

C:input(argument:data_type)
BEGIN -- if available

END;
```

図 22: EPDL と基本モデルの対応

- クラス定義ブロック
- ファンクション (関数) ブロック
- プロセスインターフェースブロック
- プロセスアクションブロック

それぞれのブロックの開始と終了の部分の例を以下に示す．なお本言語での--の後は，言語中でのコメントとして付属している．

```
TYPE  length_measure := REAL ; -- タイプブロック

CLASS c1; -- クラスブロック
...
END_CLASS

CONSTANT  -- 定数宣言ブロック
  buffer_size : INTEGER := 256 ;
END_CONST ;

FUNCTION func_example( ) LOGICAL ;
...       -- ファンクション(関数)ブロック
END_FUNCTION ;

INTERFACE i1; -- インターフェースブロック

END_INTERFACE;

PROCESS p1; -- プロセスインターフェースブロック
...
END_PROCESS;

p1:interrupt(); -- プロセスアクションブロック
BEGIN
...
END;
```

これらのブロックは，互いに入れ子になることはなく，またブロックの識別子は一意でなくてはならない．

表現される活動モデルの構造

図 23に設計生産プロセス記述言語によって表現される活動モデルの概要を示す．一つの活動モデルは，その活動に関与するプロセスと入出力対象の記述によって行われる．

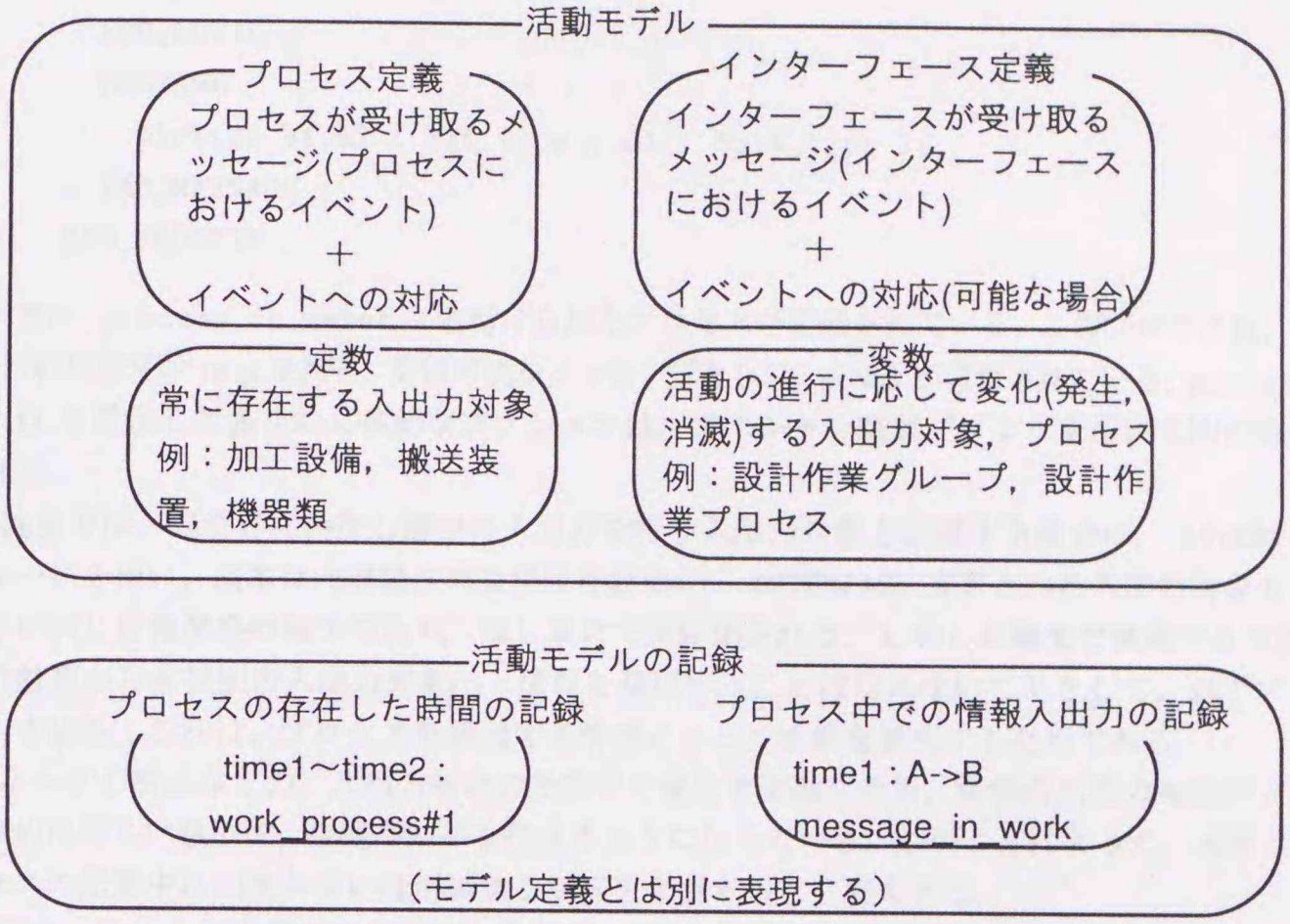


図 23: 表現される活動モデルの構造

活動中に存在するプロセスや入出力対象は，変数又は定数として宣言される．すなわち，活動中に常に存在する入出力対象 (設備機器や搬送機器，アプリケーションシステムなど) は定数として宣言される．また，活動中に生成されたり消滅してしまう入出力対象やプロセス (設計グループの阻止木構造や，設計活動プロセス) は，変数として宣言される．

活動の記録の書式そのものは，本論文の対象ではないが，活動プロセスが存在した時間的な記録と，そのプロセスの内部で交換された情報の記録によって行う．

4.2.2 プロセス及び入出力対象の入出力仕様表現

プロセスのインターフェース及び動作の記述

プロセスは，特定の活動における情報の入出力を表現するため，他のプロセスから起動され，入力を受け取り，処理を行い，結果を出力する手続きとして表現される．プロセスと入出力対象は，入出力仕様の記述では構文上の区別は行わずに表現される．

プロセス及び入出力対象への情報入力の仕様は，プロセスインターフェースブロックで記述される．プロセスインターフェースブロックでは，プロセス (入出力対象) を起動する際の引数及び局所変数，他のプロセス及び入出力対象から受け取るイベント (CCS 的な見方ではポート) 及びその際に渡される引数などによって定義される．プロセスインターフェースブロックの記述例を次に示す．

```
PROCESS process_in_robot ;
  LOCAL
```

```
    tmp : INTEGER ;
  END_LOCAL ;
  MESSAGE
    pirtt1( b1,b2 : bit_type ; d1 : byte_type );
  END_MESSAGE ;
END_PROCESS ;
```

この例では，process_in_robot と名付けられたプロセスが定義されている．このプロセスは，INTEGER 型の局所変数 tmp を持ち，受信可能なメッセージとして，pirtt1 が宣言されている．process_in_robot が pirtt1 を受信した場合の具体的なアクションは，アクション定義ブロックを用いて別の部分で宣言される．

本論文では，実世界に存在し情報の入出力を行う入出力対象を表現する場合は， INTERFACE のキーワードを用い，通常は内部動作の表現は行わない． INTERFACE 宣言された入出力対象もプロセスも伴に同じ情報交換の相手であり，同じ宣言で実体化される．しかし本論文で提案する言語では，入出力対象から直接別の入出力対象へと情報を発信することは行えないこととして，必ずプロセスを経由させる．これは，プロセスを通過する情報によって活動を表現するためである．

グループの構成などの，入出力対象の活動中の変化を表現する為，情報の入出力対象がプロセスと意味的に等しい扱いで，生成や削除を行えるようになっている．このことによって，入出力対象がプロセスの活動中に出現あるいは消滅する状況を表現することができる．

活動中，常に入出力対象が存在する場合は，それらは定数として宣言される．それ以外の途中から起動されるプロセスや入出力対象は，変数として宣言される．しかし何も存在しない状況で，いきなり入出力対象が出現することは表現できないので，外部で入出力対象が出現したことをイベントとして受け付けることができるプロセス又は入出力対象が必要になる．

この拡張でプロセスと入出力対象は，手続き的な表現から，プロセスが受け付けることができるイベントと，それに対する反応の組み合せとして，次の項目で表現する入出力インタフェース定義として扱われる．

- プロセス及び各入出力対象の間で，どのような入出力現象をイベントとして扱うか
- そのイベントで，どのような情報が交換されるか
- プロセスの内部変数として，どのような情報が貯えられるか
- 入出力のイベントによって，内部変数がどのように変化し，他に対してどのような出力が行われるか

メッセージを受信した際のプロセスの具体的なアクションは，アクション定義ブロックを用いて宣言する．プロセスアクションブロックの記述例を次に示す．

```
PROCESS process_robot ;
  MESSAGE
    move_to_home_position( );
        :
  END_MESSAGE ;
END_PROCESS ;
```

```
process_in_robot : move_to_home_position( ) ;
LOCAL
 tmp : INTEGER ;
END_LOCAL ;
BEGIN
 ....
END ;
```

この例では, process_robot と名付けられたインターフェースが定義され, 受信可能なメッセージとして, move_to_home_position( ) が宣言されている. process_robot がmove_to_home_position( ) を受信した場合の具体的なアクションが次のブロックで定義されている.

複数プロセスの起動及び並行動作の表現

3.1.2で挙げた, 並列挙動, 親子関係などの活動の構造化, 活動同士の時間的相互関係を識別子として取り扱うために, 以下の状況を記述できる能力を持たせた.

- 一つのプロセスから複数のプロセスが起動され, それぞれが並行して動作する状況
- プロセス及び対象間での情報の入出力
- 特定時間動作が遅延する状況

並列実行するプロセスを記述するために, EPDL は Concurrent C 言語 [GR89] [ナイ 89] と同じくプロセス型のデータ型を用いる. プロセス型の値は, 同じタイプの複数のプロセスを識別するために用いられる. この値を引数とし, そのプロセスが活動中, 又は停止しているか, 等の情報を返す関数を用意する.

次に示す記述例に従って, 子プロセスの起動に関して述べる.

```
LOCAL
 self, P1 : PROCESS ;
BEGIN
 self := SELF_PROCESS();
 P1 := chield_process( self );
  :
END;
```

ここで, chield_process は, 起動の際にプロセスタイプの変数一つを引き数として渡されると宣言されているものと仮定する. この例では, 起動する親のプロセスが自分のプロセス ID を引き数として与えている. 起動された chield_process のプロセス型の値 (すなわちプロセスの一意な識別子) が, 返されて P1 に代入される. 起動されたプロセスは, 親プロセスと並行に活動して情報の入出力を行う. このような起動の表現は, 3.2.2で CCS に基づいて表現したプロセスマネージャの機能によって実現される.

プロセスは, 起動と初期化の後にイベントループに入り, 受け付けたイベントごとに定められた対応を取る.

他のプロセス又は入出力対象への出力

あるプロセスから他のプロセス及び入出力対象への入出力は，相手のID値，そのプロセスに対して定義されたイベント名，及び受け渡す値を組み合わせたイベントコール文を用いる．図では子プロセスのp1に対してsend_valueで値の出力を，他のプロセスからget_valueで値の入力を受け取っている．

```
LOCAL
 self, P1 : PROCESS ;
 tmp : INTEGER ;
 result : LOGICAL ;
BEGIN
 self := SELF_PROCESS( self );
 result := P1.child_process_message( tmp );
```

この例では，起動する親のプロセスが自分のプロセスＩＤを得た後で，プロセスP1に対してtmpを引き数としたメッセージchild_process_messageが送られている．

ブロードキャストメッセージ

通常，入出力が1対1で行われるのに対して，1対多の関係で複数のプロセスに入力できる機能を設ける．あるプロセス又は入出力対象のインターフェースブロックで，メッセージに予約語BRODADCASTを付加して宣言することで，そのタイプの全てのプロセスで同じ入力を受け付けることができる．この機能は，3.2.2で述べた同一種類のプロセスへの同報機能の言語仕様としての実現である．

ブロードキャストメッセージの例は，次のクロックインターフェースの例と合わせて示す．

一定時間ごとの動作

上記の1対多の入力として，一定時間ごとに発せられるメッセージを受け付けるインタフェースを定義する．設計生産活動モデルでは，この機能を継承して具体的な動作を再定義することによって，一定時間毎，又は特定の時間における動作を定義する．

プライオリティ

プロセスへの情報の到着やプロセス内部での処理は瞬時に行われるとしているが，例えばdelay文で実行を遅延させている場合に，他からの情報の入力が起こることが考えられる．メッセージに対する処理の優先度を定義するために用いる．ある動作を実行中に別のメッセージが入った場合，大きなプライオリティ値を持っているメッセージが優先される．場合によっては，動作中の処理を中断して，後から来たメッセージの処理を行う．

```
PROCESS clock_interface ;
  MESSAGE
    BROADCAST every_one_hour( current_time : time ):300;
    BROADCAST every_one_minute( current_time : time ):200;
    BROADCAST every_one_second( current_time : time ):100;
  END_MESSAGE ;
```

```
END_PROCESS ;
```

この例では，ブロードキャストメッセージとクロックインターフェースの定義の例としてを示している．各メッセージにはプライオリティ値が付けられている．この例では特定の時間に，
BROADCASTevery_one_hour(current_time : time):300; と
BROADCAST every_one_minute(current_time : time):200; との2つのメッセージが到着した時には1時間間隔の every_one_hour の方が優先される．

各メッセージはブロードキャスト宣言された入出力仕様を継承する．各設計生産活動モデルでは，継承した入出力仕様を再定義することによって一定時間ごとの具体的な動作を定義する．

```
PROCESS timer ;
  INHERIT_FROM ( clock_interface );
  MESSAGE
    every_one_hour( current_time : time );
  END_MESSAGE ;
END_PROCESS ;

timer : every_one_year
            ( current_time : time ) ;
BEGIN
  IF ( current_time == X-TIME ) THEN
      ; -- Do
END ;
```

インタフェースの宣言，継承，ブロードキャスト，及びプライオリティの例を示す．まずclock_interfaceとして，一定時間ごとのブロードキャストメッセージを受け取るインタフェースが定義される．この例は，クロックインターフェースからの継承と，一定時間ごと動作の再定義を示している．

この定義内容をtimer が継承し，1時間ごとのメッセージevery_one_hour を受け取ることが可能になる．このイベントに対する動作とプライオリティを再定義することで，1時間ごとのメッセージを受け取り，受け取った時間を確認して特定の時間になった時に，規定された動作を行う．

4.2.3 その他の言語仕様

データ型の記述

EPDL は，数値及び文字列などに関する基本的なデータ型，単純継承を用いるクラス型を備えている．この言語は，入出力対象及びプロセスの記述を目的としているので，プロセス型の特殊なデータ型の表現を持つ．さらに，データモデルの構造に関する情報や実際のデータも入出力対象としたが，データ型の操作などの機能は言語の仕様としては組み込まれていない．これは，データモデルなどの内部構造に依存することを避けるためで，それらの対象間で入出力される情報は，基本的なデータの組み合せによって定義する．

継承機能

他のプロセス及び入出力対象に対して定義された入出力仕様を多重継承して再定義する機能を備えた．入出力対象やデータベースなどの種々の入出力対象で，共通に必要とされる特性を定義してお

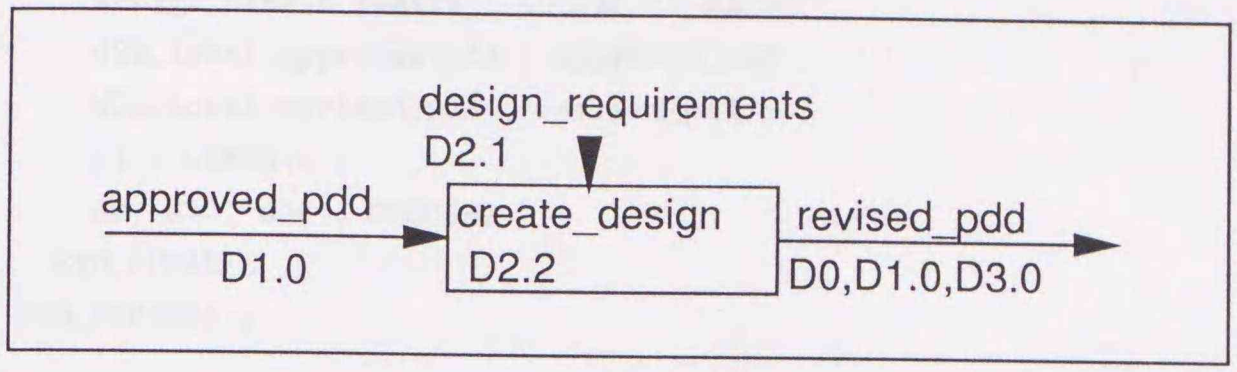


図 24: EPDL による表現例

く．次の第5章で述べる特定の活動モデルでのインタフェースやプロセス，活動スケジュールのデータベースなどは，予め定義した仕様を継承して再定義することでモデル化する．

4.2.4　モデル化の例

SADT モデルに対して EPDL を用いた記述例

本論文の目的は，SADT 記法のモデルの記述表現ではないが，単純な入出力の記述例として，機械設計活動における設計変更過程に関する活動情報を一般企業から提供してもらい，内容の記述を試みた．子プロセスへの分解，プロセス名及びデータ名などの用語は，STEP の製図データ作成手順モデル [ISO95d] に準拠して行なった．図 24に IDEF-0 によるモデルの表記の一部を示す．この例は，設計データを変更する活動を一つのプロジェクトとした中の，d22_create_design と名付けられたプロセスに関する記述である．より具体的には，以前の設計活動で既に承認されている設計データapproved_pdd を入力として，参照した設計要求design_requirements に従って変更し，修正結果revised_pdd として出力する活動の部分をモデル化している．モデルとして表現するには，幾つかの手法が考えられるが，本論文では IDEF-0 で表現された一つの活動を一つのプロセスとして表現した．

このモデルを本論文で提案した EPDL で記述すると次のようになる．

```
PROCESS design_process ( parent : PROCESS ) ;
   LOCAL
      self : PROCESS ;
   END_LOCAL ;
END_PROCESS ;

PROCESS d22_create_design( parent : PROCESS ) ;
  INHERIT_FROM ( design_process );
  MESSAGE
    d22_input( d22approved_pdd : approved_pdd );
    receive_design_rquest( d22form_design_request : design_request);
  END_MESSAGE ;
  LOCAL
```

```
    design_create_status : LOGIAL := FALSE ;
    d22_local_approved_pdd : approved_pdd ;
    d22_local_revised_pdd : revised_pdd ;
    r1 : GENERIC ;
    d0, d10, d30 : PROCESS
  END_LOCAL ;
END_PROCESS ;

d22_create_design : d22_input( d22approved_pdd : approved_pdd );
BEGIN
    d22_local_approved_pdd :=  d22approved_pdd ;
    design_create_status := TRUE ;
    db.send_design_request();
END ;

d22_create_design : receive_design_rquest
        ( d22form_design_request : design_requst );
BEGIN
    IF ( design_create_status == TRUE ) BEGIN
        : -- 内部アクション
        ( d0, d10, d30 ).revised_pdd_created( d22_revised_pdd );
        design_create_status := FALSE ;
    END ;
END ;
```

ここで記述しているモデルでは，初めに実際にモデル化されるべきプロセスの上位のプロセス PROCESS design_process (parent : PROCESS) ;がモデル化され，プロセスが起動される際に渡される引き数parent，局所変数self が宣言される．

次に実際に表現するプロセスd22_create_design が記述される．プロセスが起動される際に渡される引き数parent 及び他のプロセスから情報を受け取るためのインターフェース設計変更の要求 d22_input(d22approved_pdd : approved_pdd); と，設計変更に関する要求データの受け取る receive_design_rquest(d22form_design_request : design_request); とが定義される．プロセスd22_create_design は起動された段階で 2 つのメッセージを受信することができる．

プロセスのアクションは，この 2 つのメッセージが到着した際のリアクションとしてモデル化されている．ただし 2 つのメッセージの内，設計変更を行っている状態でない場合には設計変更に関する要求事項を記述したデータの受け取り receive_design_rquest(d22form_design_request : design_request); のメッセージが送られても無視されるように，設計活動の変更状態にあるかどうかを示す内部変数design_create_status を初期値をFALSE として宣言している．この状態変数は，メッセージd22_input が到着した時にTRUE となり，同時にプロセスは設計変更に関する要求事項を記述するデータを受け取るためのメッセージをデータベース (仮にdb としている) に対して発信する．内部変数design_create_status == TRUE の場合に，設計変更に関する要求事項を記述するデータがプロセスに届くと実際の設計変更作業が

行われる．ただしこのモデルでは，実際の活動に関する部分はコメントアウトされて表現されていない．本来は，その活動を担当する設計活動者との間で情報の入出力が行われる必要がある．このモデルでは，プロセスd22_create_design は変更データが完成した段階で，それを対応するプロセスに対して発信して内部変数design_create_status をFALSE に戻して最初の開始状態に戻る．

設計生産プロセス記述言語の仕様に関する考察

SADT 手法によるモデルは，本来は動的なプロセス表現が完全には行えない点が指摘されている．従って 4.2.4での記述例は，手続的な活動過程の表現例であり，並列活動状態の表現には至っていない．本論文の目的は，SADT 手法によるモデルの表現ではない．しかし本論文で提案している基本モデルに基づいた設計生産プロセス記述言語による記述では，以下の項目で不都合な点が見つかり，第5以降で提案するモデルの課題となると考えられる．

1. プロセスの起動のタイミング：SADT 手法において四角で示される活動は，具体的な活動というよりも情報の処理の手順としての記述に重点が置かれている．従ってこの手法に基づく元々の資料 [ISO95d] では，各プロセスを起動するタイミングに関しては何の記述もなされていない．即ち，特定の入力が起こった際に活動として処理される形でしか表現されておらず，各活動が時間軸上でどのように協調するかに関しての記述が行われていない．本論文でモデルを構築する際には，予め全てのプロセスが起動されている前提でモデル化を行う必要があった．

2. 入力メッセージ：SADT 手法では，特定の活動に対する入力は，四角の左側から入力される一般的な入力データと，四角の上側から入力される参照データが入力される．これらの入力は，メッセージではなく入力されるデータの型のみが指定されている．通常は複数の値を引数として意味を持を持たせてメッセージとして入力させる必要があると考えられる．従ってモデリングの際には，データ以外にどういうメッセージとして入力されるかを新たに指定する必要があった．

   順序だてた情報入力の必要がある場合，例えば，あるメッセージを受け取った後で外部からの情報の入力が必要な場合，SADT 手法では表現することが出来ない．本論文のモデルでは，プロセスの内部で状態遷移を示す変数を持たせて対応することができる．

3. 概念的モデル：今回行った記述は，実際のアルゴリズム及び送受されるデータの情報が不足したり，データベース及び作業者などの入出力対象に関する定義を省略した概念的なモデルである．従って実際のインプリメントを考えると，未定義の部分をより詳細に定義する必要があると考えられる．

4. プロセスの構造化：複雑な並行活動を表現するためには，プロセス及び入出力対象を階層構造などを用いて構造化する必要がある．

5. 入出力対象の変化：今回の記述例では，各入出力対象は，プロセスの実行が進む間に変化しない定数として表現したが，実際の設計活動の組織構造が変化したり，新たに加わるような状況を表現する必要がある．

4.3 特徴及び他の言語との比較

4.3.1 本論文における設計生産プロセス記述言語の特徴

本論文で提案した設計生産プロセス記述言語の特徴を，4.1.2 でまとめた問題点の項目に対して述べる．

1. 言語仕様全体としての課題

   - 表現要素の種類：4.1.2では言語の表現要素を増加させることによる表現能力の向上が，結果的に記述内容が複雑なモデルとなってしまい，直感的に理解することが難しくなる点を指摘した．本論文で提案した設計生産プロセス記述言語では，この点に注意して，入出力対象とプロセスを構文上の区別を行わずに表現することとした．またイベントドリブン方式を用い，プロセスのインターフェースと個別のリアクションの記述を別々に行うこととして，各ブロックの記述を独立して行うことを可能にした．
   - 構造化しやすい仕様：制約指向言語を用いた場合には，複数の制約が結果的にどういった影響を及ぼすかを容易に把握できるかどうかが問題点となる．本論文で提案した言語では，入出力の仕様は多重継承構造を用いることによって，特定の目的ごとの入力と，その入力に対する反応をモデル化したうえで，組み合わせることを可能としている．これは，目的ごとのメッセージと反応は，基本的には排他的な関係にあり，相互に関与することは少ないとの見方から行っている．従って表現されたモデルが理解しやすいかどうかは，この階層構造をどのように構築するかが問題となる．
   - EXPRESS-P では，他のエンティティの共有や，参照などのエンティティ同士の関係とシグナルを発信又は受信するための交信の構造が 2 重に定義されることへの複雑化の懸念が上げられた．本論文では，このような相互参照の複雑化は避けて基本的には入出力の関係しか対象としていない．また，予め通信構造を定義しておくのではなく，入出力の対象や交信相手のプロセスをプロセス型の変数として保持することで通信を行う．従って活動が進行する間に動的に変化する通信構造が可能になる．

2. プロセスの表現

   - 4.1.2では，プロセスを独立した存在として表現し，プロセス相互の関係を表現する必要が指摘されている．本論文で提案した設計生産プロセス記述言語では，入出力対象同士の情報の交換を禁じており全ての情報は一旦あるプロセスに入力された後に何等かの形で別のプロセスや入出力対象へと出力される．
   - 入力を受け付けるためのステートメントで受信を陽に記述して入力を待つ方式では，タイミングの取り方の記述が難しくなることが考えられる．このため，本論文ではプロセスに対する情報の入力が行われた場合に，その入力に対する反応を取る方式で表現する．基本的にはプロセスの反応動作は瞬時に行われることとし，動作時間を伸ばす場合には，時間待ち用のステートメントを用いて表現する．プロセスの活動を手続き的に行う場合でも，必然的にイベントループを表現する必要が生じるがイベントドリブン方式を用いることによってイベントループの実現をシステム側に任せて，モデルを構築する側では，より重要な入力メッセージと対する反応のモデリングに傾注することができる．

3. 設計生産活動に関与する対象の表現

- これまで考察を行った設計生産プロセス記述言語では，基本的には静的な対象間での活動を対象としており，活動の主体の動的な構造変化までは表現しきれていないため，入出力対象の構造が動的に組み替えられる状況が考えられ，このような活動を表現する必要があった．本論文で提案した，設計生産プロセス記述言語ではプロセスの起動を別のプロセスから行うことができるので，例えば設計活動者が変わらなくても，特定の活動に関するプロセスが起動され，そのプロセスが対象とする作業者のグループが変更されることで，設計活動グループの構造の変更を表現することができる．

4. 入出力される情報やリソースの表現

- 本論文の目的である設計生産における活動を表現するためには，送受される命令は，特定の種類のシグナルのような情報を言語の仕様として予め決めておくのではなく，ユーザが規定できるようにする必要がある点が指摘された．本論文で提案した設計生産プロセス記述言語では，入出力さえる情報は，基本的な整数や文字列型の変数を内部に持ったクラスのインスタンスを入出力することで表現する．
- 実際の生産システムにおける加工プロセスで，供給されたり消費されたりする各種のリソースを表現する必要が指摘された．本論文では基本モデルとして情報の入出力を活動として表現しているので，被加工物の加工や物の搬送などはモデル化の手法で対応する必要がある．

5. 物理的な表現や実装への配慮

- 本論文ではEXPRESS-PにおけるSTEP物理ファイル上での表現のように，特定の時間における通信構造を特定のフォーマットに示す方式は用いずに，システムとしての挙動の表現に重点を置いている．これは，本論文の目的では，特定時間の状況の表現ではなく，システムの状態遷移の方式を表現して，プロセスに対する情報の入出力の観測を記録することで，そのシステムの活動の記録とすることに主眼をおいているからである．
- EXPRESS-Pにおける物理ファイル上での参照構造の実現は，例えばSELECTステートメントの解釈に関して不安定な部分が見受けられるため，通信構造の定義に関しても同様の問題を持つ可能性がある．本論文では，全体としてのモデルの参照構造と，情報や物の流れの構造とを二重に定義する構造は，取らないこととする．

4.3.2 他の言語との比較

本節では，4.1.1で調査した研究と本論文との比較を4.1.2でまとめた問題点に関してまとめる．比較する項目は，4.1.2で上げられた次の項目

1. 表現要素の種類と多数のプロセス間の関係を表現する際の複雑化の懸念
2. 通信構造の動的な変化
3. 活動中のアクティビティ(又はプロセス)の陽な表現による活動相互の関係の表現
4. 入力ステートメントを用いない情報の受信表現
5. 入出力される情報の高度な表現及びユーザによる定義

6. 生産活動における製品など物の移動や加工作業などの表現

7. 入出力構造の表現方法

とした．比較対象の研究として，4.1.1で調査したFDL，CIMOSA，COSL，EXPRESS-P，EPDLに関して行う．

1. FDL では，本論文の目的の設計生産活動の表現とはかけ離れた，対象とする部品のパラメータ間の制約を定義することを目的としている．制約の表現を主目的としているため，特に複数の設計要素相互の関係を表現する際，記述が複雑化してしまい，記述された意味を容易に把握できないことが懸念される．この言語ではシステムの動作としての視点はないので，通信構造の動的な変化やプロセスを陽に表現して相互の関係を記述することは不可能である．また，入出力の状況を表現することはできない．

2. CIMOSA では，言語仕様として言語要素を豊富に用意して表現能力を高めている．従って言語の仕様として記述された内容の意味的な一貫性を保つことが難しくなると考えられる．CIMOSAで用いているMSS言語では，CIMの設備仕様を表現することを目的としているために，表現されるプロセスはCIMにおける機器又は装置に該当するドメイン内部でだけ実行される．本論文で提案した特定の活動に関する情報を伝達する媒体としての役割を持たせることは難しいと考えられる．このため，活動中のアクティビティ(又はプロセス)を陽に表現して活動相互の関係を記述することはできない．情報の入力は，入力ステートメントを用いないイベントドリブン方式となっている．また基本的にはCIM設備間で受け渡される情報だけ表現できればよいが通常のプログラミングで用いられるレベルの整数，実数などの数値表現や文字列データなどを入力とすることはできる．生産活動における製品などの物の移動や加工作業などを入力リソースとし表現できるようになっている．しかし，設計活動グループの組織変更のような動的な変化は表現できない．

3. COSL は，低レベルでの入力と反応の表現を目的としており，入力される情報の種類は言語仕様として予め決められている．従って，入出力される情報の高度な表現及びユーザによる定義を行うことは出来ない．この言語では，入出力される情報までも言語仕様として決められているので，表現要素の種類としては非常に少ない．COSL では，基本的な入出力関係しか表現できないので活動中のアクティビティ(又はプロセス)の陽な表現による活動相互の関係の表現や，多数のプロセス間の関係を表現する際の複雑化は，起こらないと考えられる．また基本的には生産設備の機器間の通信の表現を目的としているので，通信構造の動的な変化を表現することはできない．情報の入力の表現は，言語仕様として予め定められた入力コマンドを用いて，入力ステートメントを用いない形で行われている．生産活動における製品など物の移動や加工作業などの表現などを行うことも出来ない．

4. EXPRESS-P では，STEP で事物の表現を意味するエンティティの内部でプロセスを実行することにして，実際の通信構造は物理ファイル上で表現を行う．従って，データとしての参照構造と，情報入出力の構造とが交錯してデータ構造が複雑化することが考えられる．さらに通信構造の動的な変化や活動中のアクティビティ(又はプロセス)の陽な表現による活動相互の関係の表現を行うことは出来ない．情報の受信表現は，入力ステートメントを用いる方式で表現し，入力をイベントとするイベントループを，ユーザが陽に設定しなければならない．入出力される情報の高度な表現及びユーザによる定義は，STEP の EXPRESS におけるエンティティをそ

Subject	FDL	CIMOSA	COSL	EXPRESS-P	EPDL
Variation of language lexical element	×	△	○	△	○
Dynamic change of communication structure	×	×	×	×	○
Explicitly process representation	×	○	△	○	○
Using an input statement	×	○	○	×	○
Representation of input/output data	×	○	×	○	○
Representation of resource for manufacturing	×	○	×	△	○
Representation of input/output structure	×	○	×	×	○

表 8: 言語仕様の観点から見た既存の研究及び本論文の評価
○ : Completely support, △ : Not completely, × : Not support

のまま入出力としているので，表現能力には問題はないと考えられる．生産活動における製品など物の移動や加工作業などの表現は，STEP データを入出力させるプロセスとしては表現することができる．

5. 本論文で提案した EPDL では，活動の表現に主目的を置いているために，言語仕様としてできるだけ少ない表現要素で記述できることを目的とした．

   また要素間の関係は，活動中の入出力対象を変数として持つ以外には定義していないため，データとしての参照や入出力対象としての参照が交錯して複雑化する懸念は少ないと考えられる．

   このように入出力先の参照構造を，予め固定的に宣言するのではなく，通信構造の動的な変化を表現することが可能である．また，並行に動作する各プロセスを陽に記述することが可能であるので，その活動相互の関係の表現を行うこともできる．

   また EPDL では，入力ステートメントを用いない方式で情報の受信表現を行っている．このために EXPRESS-P に見られるようなイベントループをユーザが自身で定義する必要はなくなる．入出力される情報は，基本的な数値型や文字列型又それらを組み合わせたユーザ定義のクラス型の値を入出力することが可能である．データ型に関する情報などの高階の情報は対象外としている．生産活動における製品など物の移動や加工作業などの表現は，基本的には情報の入出力の枠内で表現することとする．この問題については，第 5章以降の課題とする．

ここまで述べてきた内容を表 8にまとめる．4.3.1 で述べたように，本論文で提案した設計生産プロセス記述言語 (EPDL) では，これらの問題点に関しては克服することができたと言える．

4.3.3 本論文で提案された方法での表現の限界

本節では，本論文で提案した設計生産プロセス記述言語 (EPDL) での表現の限界に関して考察する．

- 本論文で提案した設計生産プロセス記述言語では，表現する対象を情報入出力の対象として表現する．この方式で表現できる範囲であれば問題はないが，逆に実際の搬送など物が移動したり変化するようなモデルでは，被加工物を内部状態を保持した入出力対象として，この対象への操作として加工プロセスをモデル化するか，内部情報を保持した情報単位として各対象とプロセスの間を入出力されることで表現することが考えられ，この点は第5章で述べるモデルの基本構造の部分で対応する必要がある．

- 本論文で提案した言語仕様では，無計画にプロセスを並行動作させた場合に各プロセスの相互の関係が表現されたモデルが複雑化してしまう懸念がある．この問題点を克服するためには，プロセスをある定式化された関係に限定することで構造化する必要がある．この点は，第5章で述べるモデルの基本構造の部分で対応する必要がある．

- 本論文で提案した言語仕様では，通信を行う構造が動的に変化する仕様となっている．この方式では，ペトリネットなどで用いられている解析手法が実装できないことが考えられる．従って，プロセス代数を用いた表現による評価方法を確立する必要があると考えられる．

- 本論文で提案した言語仕様では，特定の時刻に於ける入出力対象や動作中のプロセスのスナップショットを表現する方法を確立していない．例えばパラメトリックモデラによって作り出されたデータは，そのモデリングシステムの特定の時刻における内部状態の内，形状データなどに関する部分を表現していると考えられる．このデータは，特定の属性間での依存関係がユーザによって定義されており，モデリングシステムを通じて設計活動者からの作業入力が行われることによって変化する．このモデリングの過程を本論文での基本モデルに基づいて考えると，モデリングシステムのモデルに相当するプロセスが動作しており，設計者がパラメータの依存性を入力することによって，データの属性間での依存性 (状態遷移則) が定義され，その後特定の形状や寸法を変化させることによって，一貫性を保ったまま (決められた遷移則に沿って) 属性の値が変化する．このようなプロセスの内部状態を表現したものがパラメトリックデータとなる．このような研究を発展させるためにも，特定の時刻における名部状態を表現する技術を確立する必要がある．

- 本論文で提案した設計生産プロセス記述言語では，記述方式として，外部挙動規定法を用いているため状態遷移システムの表現を陽に行う仕様にはなっていない．従って，特定の入力の組み合わせが連続しておこった際に，プロセスの内部状態がどのように遷移していくかを表現するためには，プッシュダウンオートマトンのような状態遷移機械となる関数を，ユーザがプログラミングして表現しなくてはならない．

- 本論文で提案した言語では，入力を受け付けるためのステートメントやイベントループをユーザが陽に記述する必要はなく，言語仕様として予め組み込まれている．このような仕様での複数の動作の同期は，4.2.3の所で上げたクロックインターフェースを用いた特定の時刻での動作として表現できる．

4.4 結 論

第4章では，第3章で提案した基本モデルに基づいて設計生産の活動を表現するため，形式仕様記述言語を用いた既存の研究に関する分析を行い問題点の抽出を行った．得られた問題点を解消するために設計生産活動を表現するための設計生産プロセス記述言語の提案を行い以下の結論を得た．

- 設計生産活動を形式仕様記述言語を用いて表現する既存の研究に関する分析を行い，これまで用いられてきた言語仕様の分析を行った．
- 本論文で目的としている設計生産活動を表現するためには，次の項目が必要であることが判明した．
  - 言語要素を少なくして仕様の複雑化を避けること，
  - 独立した動的なプロセスを表現し，相互の強調を表現すること．
  - 入力ステートメントを用いないで，イベントループを言語仕様として表現すること．
  - 入出力されるデータをユーザが規定できるようにすること．
  - 生産過程での入出力に製品などの物流を表現すること．
  - 予め定義されて変化しない通信構造を用いない．
  - データとしての参照と情報入出力の構造が交錯してモデルが複雑化を避けること．
- 本論文で設計生産に関する活動を表現するための設計生産プロセス記述言語の提案を行った．これは，3.2.1で提案した基本モデルに基づくもので，次の情報の記述を行う．
  - 設計生産の活動における種々の対象の間で特定の活動に関する情報をやり取りする概念的な媒体としてプロセスを表現する．
  - またそのような活動に関与する様々な対象を情報入出力の対象として表現する．
  - 入出力される情報が，どのような種類のものであるかの型に関する規定．
- 設計生産プロセス記述言語の特徴は，イベントドリブン方式を用いているので，ユーザがイベントループを記述する必要がないこと，同じ理由からプロセスの動作を，メッセージ毎の形で表現できること，これによってユーザは，入出力されるメッセージの表現に傾注できること，などがある．
- 4.1.2 でまとめた課題に対して他の言語との比較を行い，本論文で提案した基本モデルが有効であることを明らかにした．
- 本論文で提案した言語仕様では情報以外の物の移動を直接表現することが難しいため物流の表現に関しては，
  - 移動する物体を情報として各対象の間を入出力させる．
  - 移動物体も入出力対象として表現する．

  の２つの方法で対応できることを示した．
- 本論文での課題として，特定時間における通信構造の表現を行なう必要性が考察された．また発展的な研究として，パラメトリック形状モデラの研究に結び付けられることを述べた．

5　設計生産活動統合モデルの構築

1.2では，設計生産の活動を表現して活用を計ることの問題点として，本論文での対象が大規模なモデルとなる可能性があり，モデルの構築方法が重要な課題となる点に関して述べた．また2.3において，設計生産における活動を表現する大規模なモデルの構築に関する問題点の分析を行った．本章ではこれらの問題点を考慮し，第3章及び第4章で提案した基本モデルと形式仕様記述言語に基づいた設計生産活動モデルの基本構造を提案し，その基本構造に基づいた設計生産活動モデルの構築を行う．

本章の構成は，5.1において本論文のモデリングの対象として考えられる設計グループ活動モデルや生産活動モデルに関する調査を行い，構成要素や特徴，全体としての留意点などに関して述べる．さらに既存の研究における分析を行い，解決すべき具体的な問題点を明らかにする．次に5.2において，第3章及び第4章で提案した基本モデルと形式仕様記述言語に基づいた活動モデルの基本構造を提案する．5.3においてこの基本的な構造に基づいた設計活動と生産活動に間する設計生産活動モデルを構築する．最後に5.4において，提案したモデルと従来からの研究とを比較してモデルの特徴及び限界を明らかにする．

5.1　設計生産活動の全体を表現するモデル構造に関する分析

2.3でまとたように，設計生産活動の表現と活用を行うためには，組み合わされる各モデルの機能を明確にして部品化し，それらを組み合わせる全体的な構造を確立する必要があると考えられる．

本節では，まず5.1.1において対象として考えられるモデルの範囲を明確にし，5.1.2で本論文でモデル化の対象となる活動モデルに関して構成要素，活動の特徴などに関してまとめ，モデル全体として留意する点に関して述べる．5.1.3では，モデル構造に関する既存の研究報告に関して考察を行う．それらの分析に基づいて，5.1.4において本論文で解決すべき具体的な問題点を明らかにし本論文における方針を定める．なお，ここでは各種の活動計画を立案するアルゴリズミックな部分に関しては，すでに多数の研究が報告されており，本論文で目的としているものとは異なっているために範囲外とする．

5.1.1　対象モデルの範囲

本節では，対象として考えられる設計活動に関しての調査を行う．

Cleetus による仮想設計チーム

Cleetus は，コンカレントエンジニアリングへの回答として仮想チームによる設計活動を提案している [Cle93]．仮想チームは，ネットワーク上に実現されるもので，(原文では) データの中心的な蓄積を共有しなくてはならない．この仮想チームでのプロジェクトリーダーのためには，タスク，ゴール，デッドラインなどに関する連絡を可能にし，チームメンバーによる活動の進展をフォローし，個別の決定の効果をチーム全体に対して公表することを可能にする統制用の枠組みが重要となる．このようなチーム全体を覆う統制によって，次の項目の利点がもたらされる．

- チームに対してその時その時の決定の一貫性を与える．
- 従って相互の理解を与えるので，並行活動が固定的とならない．

- 視点の相違から決定に対して合意できないような事態とはならない.

この方式で定義された仮想チームは，協調して意思決定を行うためのモデルと考えることができる．なぜならばエンジニアリングあるいは製品開発のためのモデルとして特殊なものは見られない．また多重化された視点と，分離した個々の貢献からの解を得るための創造作業の同期とを要求する全ての決定作業は，製品モデルなどの取り扱いのために十分なものである．コンカレントエンジニアリングによって記述あるいはモデル化されるプロセスは，現実的には意思決定の問題であり製品開発の問題ではない，そしてそれ故に，企業全体で参加することに価値のある物となる．

この文献による基本的なワークフローでは，設計活動全体を見渡す視覚的な状態から複数の task が各設計者に割り当てられ，設計活動として個別に決定がなされる．その後，各決定事項を表明し，一貫性をチェックし，ネゴシエイトした後に次の設計段階に入るものである．

社会的アクティビティモデル

渥美 [Ats93] による "Social Human Activity Model" では，グループ化された作業者の階層における製造作業の制御は，指示単位の階層として表現でき，同様に他の全ての活動は指示単位の階層として表現され，指示は解決されるべきユニークなオブジェクトとなっている．そのような指示単位は指示セルと呼ばれ，指示セルの階層構造を構築する．

ここで指示セルから指定されるエンティティは物理的な製品には限定されずに，物理的製品から情報 (ドキュメント)，サービスまでを含み，指示を受ける側は任意の順番で拒絶する権利を持っている．さらに指示セルの階層の最下位の個々の作業者は，より高い階層の指示セルとして複数の階層に同時に同じ方法で属する，さらに多重の指示セルに同時に属し，多重な役割を同時に果たしている.

AGV の状態遷移表現

一方生産関係の分野では， 浜田らは，AGV(Automated Guided Vechicle) が生産現場において移動する領域と移動状況とを木構造の遷移グラフとして表現する方式を提案している [HM93]．この方式では，グラフの節が生産現場において物資が必ず通る交差点のような特定の地点を示し，グラフの弧は各状態の遷移を示す．このように状態遷移器械として表現できる場合には，これらの輸送機械も本論文における入出力対象として表現することが可能になる.

仮想生産システム

木村によって提唱された仮想生産システム [Kim93] は，本論文における設計生産活動のモデル表現が完全に行われ，計算機システム内部で動作させることができた場合の，一つの実装と捉えることができる．その中で現有の生産システムが抱えており，なおかつ仮想生産システムが対応しなくてはならない問題点として，次の項目が上げられている.

- 変更に対するフレキシビリティ：マーケットのニーズ，製造技術，人的要求と社会環境，などなどは急速にかつ突然変更されるものである．真にスリムな製造は，これらの変更への的確な対応が要求される.

- 人間中心の組織：高品質生産を実現するためには，自動化を達成するための人間の排斥ではなく，人間の能力の増大と人間の統合化が要求される.

- クリーンな製造：製造活動の副産物は環境問題として許されるものではなく，そのような副産物を計測し制御するための音響工学が開発される必要がある．

上の，外的環境に適応したフレキシビリティは，非常に重要な示唆を与える．すなわち，仮想生産システムそのものは，実際の生産を行う前に全ての製造活動を計算機システム内で実行させることであるが，この概念とは別に，工場内部にある特定の製造設備に捕われずに，複数の製造設備にまたがったフレキシブルな製造を考えることができる．このような仮想工場と考えた場合には，設備の編成やグループ化は，要求に応じてフレキシブルに行う事が可能になると考えられる．そして，それらを組み合わせた仮想的な工場の中での物資の移動状況の表現が重要になると考えられる．

本論文の目的は，仮想チームや仮想工場あるいは仮想生産の実現ではないが，これら個々の研究において要求される活動表現は，本論文における重要な課題であり，逆に上記の仮想設計生産に関する研究は，本論文から派生して実現できるものと捉えることができる．

5.1.2 対象モデルの特徴

これまでに調査したモデルの特徴を以下にまとめる．

- 設計活動に関するモデルの特徴としては，次の項目の組織構成の表現や変換に対応できるモデルにする必要がある．

  1. 設計活動では，関与する対象としては，設計者，設計者のグループ，グループの管理者，データを入出力する CAD システム，データを貯蔵するデータベースなどが考えられる．
  2. 活動，活動のゴール，デッドラインなどに関する連絡を可能にし，チームメンバーによる活動の進展をフォローし，個別の決定の効果をチーム全体に対して公表することを可能にする必要がある．
  3. この対象による活動として，通常の設計データの入力や修正の活動，データが完成した後での公表，承認活動が考えられる．
  4. 設計者グループは一般的には製品の構造によって，機械系，電装系などに分けられ，その中では部品の階層構造に応じた階層構造を取っている．
  5. 設計者グループは，常に決まった組織で運営されている訳ではなく，その組織構造は仮想的なものであり，設計活動の進展に応じて組織構成が変更されると考えられる．一つのプロジェクトに複数の作業者が関係することは無論のこと，個人の設計者が複数の設計プロジェクトに参加することも考えられる．
  6. 設計データの参照，設計データの修正変更，設計グループ構造などに関する種々の権利を設定する必要がある．
  7. 活動 (各設計プロセス) は，階層化，構造化される必要がある．また相互の関係を表現する必要がある．

- 生産活動に関するモデルの特徴

  1. 生産活動に関与する対象として，ロボット及び工作機械などの生産設備，AGV やコンベアなどの輸送設備，実際に加工される被加工物や工具などを表現する必要がある．
  2. 被加工物や工具の搬送及び据え付けの表現．

3. 被加工物や工具の実際の加工の表現.

4. 工作機械やロボットなどの仮想的なグループ化の表現

が考えられる.

- モデル全体としての課題

  モデル全体としては，2.3 で上げた以下の三項目が重要であると考えられる.

  1. 拡張性
  2. 部分ごとの規格化／部品化
  3. 複数の部分モデルを組み合わせるための全体構造 (フレームワーク)

5.1.3 設計生産の活動を全体を表現するモデル構造に関する問題点

本節では，設計生産活動の表現と実装を行うための，モデルの構造化の方法に関する，既存の研究に対する考察を行う．モデルの構造化の方法論は，これまで述べた設計生産における活動を表現するための基本モデル，及び形式仕様記述言語を用いて大規模なモデルを構築する際にモデルの一貫性を維持し，構成要素を部品化して複数のモデルで共有するなどの方法が必要となる.

ここでは，言語仕様や基本モデルではなく，その言語を用いたモデルの構造と特徴に関して考察を行う．以下の研究を対象として考察を行う.

- MMS (Manufacturing Message System) [IBM]
- STEP におけるアプリケーションプロトコル [ISO95a][鈴木 93]
- IEM [MSJ]

MMS

MMS (Manufacturing Message Specification)[IBM] は，製造業における情報交換を目的として，工作機械を仮想デバイス (Virtual Device) として表現するために用いられる．その特徴は，製造設備間のデータ通信におけるサービスを提供し，コマンドの一般的な集合を提供することにあり，オープンな環境の中での独立な機器のコミュニケーションを保証する.

MMS の仕様は，VMD (Virtual Machining Device) の概念を用いる MMS デバイスと MMS サービス手続きとからなり，形式的な方式によって記述される．VMD は， MMS に関連するデバイスの共通な性質の抽象的な表現を行うためのもので，ネットワーク外部からの可視的な振る舞いを表現する．MMS は，VMD を用いて MMS サービスの効果を記述するだけで，特定の実機能の実装方法または変換については記述していない．MMS の仕様記述は，工作機械の入出力仕様と，入力されたメッセージに対して返されるサービスに関するものであり，相互の協調活動などの表現は行われていない.

MMS では，初めに基本的な入出力とサービスの定義を VMD によって行い，より特殊な機器の仕様は，その定義を継承して再定義することによって行う．この方法は，オブジェクト指向おける属性やメソッドのに継承と同じで，一般的であるが極めて有効な方法と考えられる.

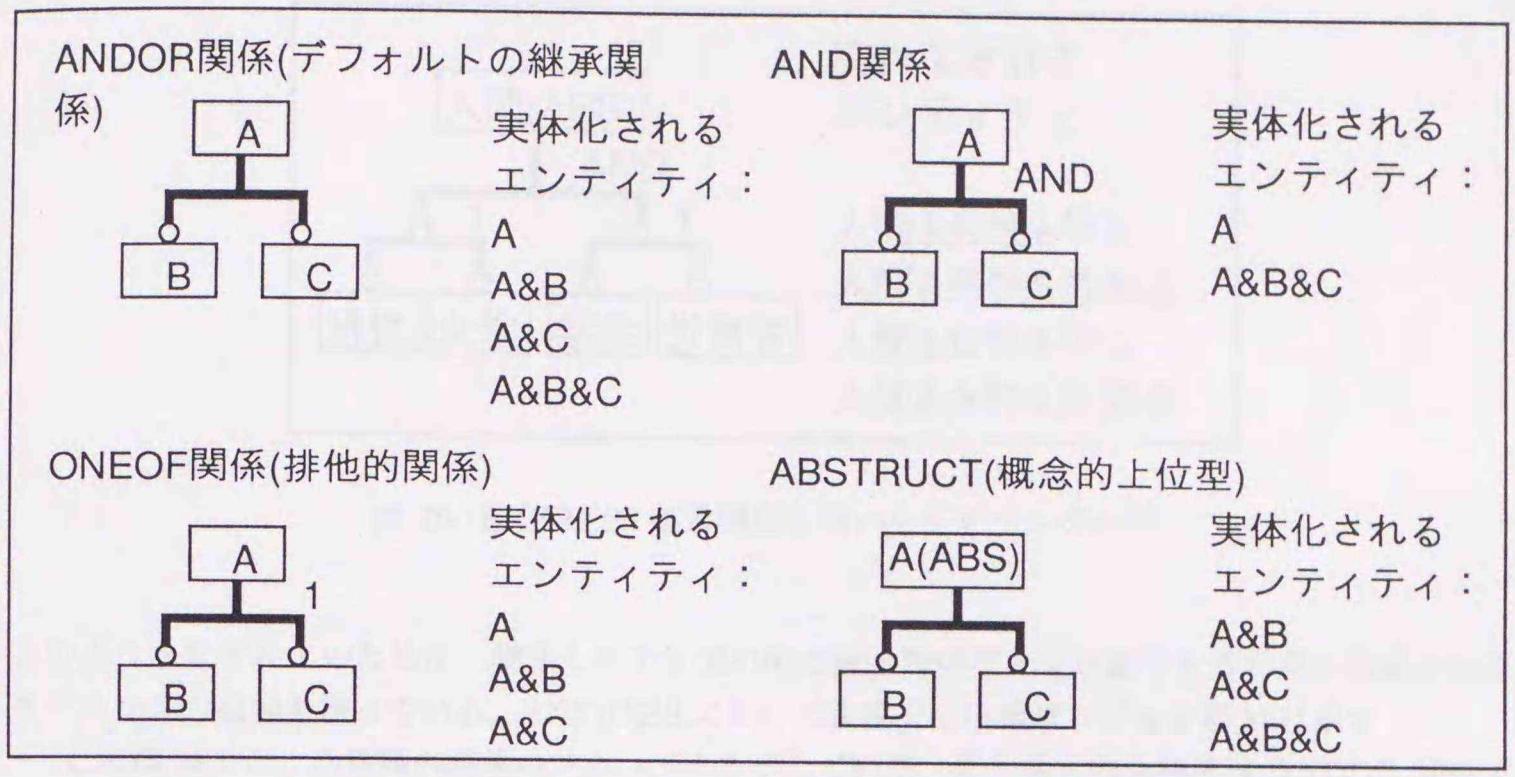


図 25: EXPRESS における継承モデル

STEP におけるアプリケーションプロトコル

複雑なモデル表現では，全体を平坦な構造で定義してしまうと一部の変更が全体に影響を与えて，信頼性を及び拡張性に欠ける恐れがある．従ってあらかじめ特定の目的と仕様を満たすようにモジュールを定義し，これを組み合わせることでモデル全体の信頼性を上げる必要がある．

STEP では，幾何や位相構造，単位系，部品構造などの基本的な要素モデルと，これらのモデルを組み合わせるための基礎的な構造とを予め定義しておき，ある分野で用いられる設計生産活動モデルを構築する場合は，基礎的な構造に基づいて基本的な要素モデルを組み合せることで，設計生産活動モデルを構築する．組み合わせられる基本モデルを統合総称リソース (Integrated Generic Resource) と呼ぶ．現在公開されている規格 (第 1 版) では，次の統合総称リソースが用いられている．

- Part 41 : Fundamentals of product description and support (製品の記述及び支援の基礎) [ISO95e]
- Part 42 : Geometric and topological representation (幾何及び位相表現) [ISO95f]
- Part 43 : Representation structures (表現構造) [ISO95g]
- Part 44 : Product structure configuration (製品構造の構成) [ISO95h]
- Part 46 : Visual presentation (視覚的表示) [ISO95i]

STEP では，これらの統合総称リソースを組み合わせることで構築された製品モデルをアプリケーションプロトコルと呼ぶ [鈴木 93].

また STEP では，統合総称リソースの記述，基本的なモデル構造の定義，アプリケーションプロトコルへの組み合せの全てを形式仕様記述言語の EXPRESS を用いて行っている．EXPRESS では，

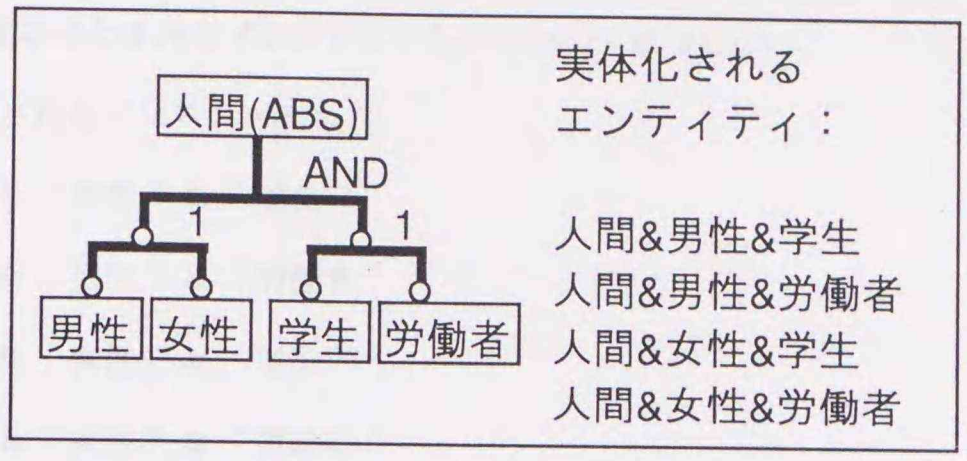


図 26: EXPRESS 継承機能を用いたモデリングの例

このようなモデル化のために，継承した下位型の間で排他関係などを定義できる非常に複雑な継承モデリングの機能を持っている．EXPRESS において表現できる継承モデルを図 25に示す．

この図 25では，四種類の継承のメカニズムを示している．最も基本的な継承は ANDOR 関係であり，他の継承を特に指定されない限りこの形式の継承が行われる．ANDOR 関係では，インスタンス化される下位型は， ANDOR として示したリストの中のエンティティの順列組み合わせとなる．図では，エンティティ A の下位型としてエンティティ B と C が宣言されている．従って，A，A & B，A & C，A & B & C のエンティティがインスタンス化されることになる．次に示されているのが AND 関係である．以降の関係は EXPRESS で記述するデータモデル定義の中で陽に宣言しなければならない．下位型が AND 関係で宣言されている場合は，そのエンティティ同士は単独で現れずに，必ず一緒にインスタンス化されなければならない．図の例では，A の下位型の B と C において，B AND C と宣言されている．この例では，A，A & B & C のエンティティがインスタンス化されることになる．次に示されているのが ONEOF 関係 (排他的関係) である．下位型が ONEOF 宣言されている場合は，そのエンティティ同士が組み合されてインスタンス化されることが制限される．即ち B & & C を含むインスタンス化が行えないことになる．図の例では，A の下位型の B と C において，ONEOF(B, C) と宣言されている．この例では，A，A & B，A & C のエンティティがインスタンス化されることになる．最後は，ABSTRUCT 宣言 (抽象上位型宣言) である．これは下位型間の関係を定義するものではなく，上位型が単独でインスタンス化されないことを宣言する．通常は，モデリングのテクニックとして，仮の上位型として宣言しておき，具体的なインスタンスとして扱われるエンティティは下位型において宣言するために用いる．

次のこの形式でのモデル化の例を図 26に示す．この図では，最も上位のエンティティの“ 人間 ”は，抽象上位型として ABSTRUCT 宣言されている．その下位型として“ 男性 ”と“ 女性 ”，及び“ 学生 ”と“ 労働者 ”をそれぞれ排他的関係として ONEOF 宣言し，それらをさらに AND 宣言している．このモデルは，次の性質を意図してモデル化されている．

- エンティティ“ 人間 ”が単独でインスタンス化されてはならない．
- エンティティ“ 男性 ”と“ 女性 ”が同時にインスタンス化されてはいけない．即ち，“ 男性 ”& “ 女性 ”を含むインスタンスが出現することを宣言する．
- エンティティ“ 学生 ”と“ 労働者 ”が同時にインスタンス化されてはいけない．即ち，“ 学生 ”& “ 労働者 ”を含むインスタンスが出現することを宣言する．

- エンティティ“男性”か“女性”のどちらかとエンティティ“学生”又は“労働者”のどちらかは必ず組み合わされてインスタンス化されなくてはならない.

結果として実現されるインスタンスは,

- “人間”&“男性”&“学生”
- “人間”&“男性”&“労働者”
- “人間”&“女性”&“学生”
- “人間”&“女性”&“労働者”

だけとなる.

本論文では, EXPRESS程の複雑な継承モデルを表現する必要性はあまり無いと考えられる. しかし, 基本的な入出力の対象及び情報入出力の過程の表現をあらかじめモデル化しておくこと, 及びこれらの要素モデルを組合わせる手法を用いる必要がある. また, これらのモデル化の方法論を形式仕様記述言語の機能に盛り込む必要がある.

EXPRESSにおけるモデル表現と物理ファイルは, そのモデルを構築する作業者や, モデルデータを作り出す設計者から見ると, 情報の入出力対象として考えることができる. すなわち設計活動を物理データへの入力作業, モデルの構築作業をデータモデル記述への入力作業として考えることができる. また, 物理ファイルの構造の規定は, 特定のデータモデルを表現したデータから, 物理的なシーケンシャルファイルへデータを書き込む手順として表現することができる.

STEPにおけるデータモデルを取り扱う方法としては, STEP内部の規格としてSDAI (Standard Data Acces Interface) が検討されている [ISO95j]. これは, STEPデータを貯えた仮想的なデータベースに対するインターフェース定義であり, データの意味に関する情報を得ることも可能であるように, データモデルの意味的な情報も入出力することが可能な規格となっている. 現在は, 基本的なモデルSDAIモデルと, それらのインターエースモデルを実際の計算機言語として表現した言語バインディング仕様の検討がなされている. SDAIモデルは, STEPのデータモデルに完全に依存した構造になっており, 他のアーキテクチャとの関係は, 特に意識されていない.

本論文では, EXPRESSにおけるモデリングの概念に捕われることなしに, 上記のデータ構造定義や物理ファイルも情報の入出力対象として表現することが必要と考えられる.

IEM

IEM (Integrated Enterprise Modelling) [MSJ] は, 生産活動におけるさまざまな活動を一個のモデルで表現することを目的としている.

このモデルの基本要素は, 次の2つのモデルから構成される.

情報モデル (Information Model) IEM情報モデル (Information Model) は, IEMにおいて入出力される対象のモデル化を目的としている, それぞれが製造モデルに対する入出力の際のオペランドとなることができる. 基本的には, 次のようなクラス階層構造からなっている.

- IEM Object Class : 製品 (Product), 発注 (Order), 資源 (Resource) などの基本的なオブジェクトの定義
- サブクラス : より具体的なオブジェクトの定義

機能モデル **(Functional Model)** 企業活動の現実を，複数の情報処理活動とそれらの論理的接続と時間的挙動により表現する．例えば，物流，材料加工などの機能は情報処理の観点から見れば，ワークプラン又はNCプログラムの実行によって表現できる．機能モデルは一般に，詳述のレベル (Level of Aggregation) から次の３つで区別される．

- 機能要素 (Function Element) :
  生産活動におけるオブジェクト処理の段階的ステップを指すLevel of Description Size ごとに次の要素を使い分けるユーザの企業モデルを構築する際にユーザから利用できるように予め用意することができる．
- 機能連鎖 (Sequence of Function) :
  機能連鎖は，一般的な定義要素が提供されており，ユーザはこの定義要素を用いて所要のモデルを記述することができる．
- 自律機能ユニット (Partially Automatic Unit) :
  機能の最終表現，全体の部分の表現，企業モデルのある領域をモデル化するのに有効であり，その決定と役割を明確にすることができる．またこの単位は，企業モデルの中で自律的になされる仕事に対する活動のスコープを表現することができる．

また，機能モデルは記述のサイズ (Level of Description Size) から次の３つのレベルで区別される．

- Objects neutral
- Functions in the modell
- All concerned objects

IEMにおける一般活動モデル (IEM Generic Activity Model) は，一つのアクティビティの内部をSADT的に詳細度レベルに応じて細分化して定義する．入力や制約などの記述の方式は，IDEF [NISb] におけるそれと類似している．また入出力オブジェクトとして前述の，製品，発注，資源が用いられる．システムの外からもしくは他の活動によって生成される，活動に対する拘束と資源には次の３種類が存在する．

- モデル活動において処理される要求 (拘束)
  一般活動モデルの表記法では，その活動の左から入力される形式で現される．
- 活動に必要な拘束
  一般活動モデルの表記法では，その活動の上から入力される形式で現される．
- 活動の実行に必要な資源
  一般活動モデルの表記法では，その活動の下から入力される形式で現される．

活動の記述は，次のレベルに分類されている．

- 行為 (Action)
- 機能 (Function)
- 活動 (Complete Activity)

IEM では，モデル化にはオブジェクト指向モデル化手法を活用しており，その理由として次の項目が上げられている．

- オブジェクトとデータが密接な関係にある．
- 属性の継承機能が必要とされている．
- クラスの概念がデータと関数により記述され，オブジェクトの挙動を表現できる．また，単一のモデルで企業の様々な側面を表現できる．
- 1 個のオブジェクトは，複数の属性によって表現でき，サブオブジェクトクラスは，継承によって作成できる．
- オブジェクトの実体化によりオブジェクトクラスの実体化へ変換できる．

IEM における特徴としては，製造モデルへの入出力となる製品などの実際の存在を製品 (Product)，発注 (Order)，資源 (Resource) とする表現を可能にしている点にある．機能表現は SADT 手法の一種と考えられ，入力と出力のみしか表現されておらず，加工などのプロセスの内部表現は完全には行われていない．

本論文としては，初めに基本的なモデル構造を確定しておき，その構造に従った設計生産活動モデルの構築を行うこと．及び基本的な機能モデルを予め定義しておき，その内容を継承することによって個別の要素をモデル化する手法を用いる必要があると考えられる．

5.1.4 本論文におけるモデル構造への課題

前節 5.1.3で行った設計生産活動のモデルの構造化の方法に関して，既存の研究に対する考察からモデル構造として解決すべき課題を整理する．問題となったのは，主にグループ化／モジュール化，並行活動の表現，継承，部品など一般的な製品の入出力としての表現などであった．以下に問題点の整理を行う．

- 全体構造 (フレームワーク)
  - 一定のモデル構造を確定しておき，その構造に従って要素モデルを組み合わせてモデルの構築を行う手法を用いる必要がある．
- 複数の部分モデルを組み合わせるための機能
  複雑なモデルを構築する全体構造として，次の項目の機能が必要となる．
  - 情報入出力の構造，対象，及び過程に関する基本的なモデルを予め用意しておき，その内容を継承することによって個別の設計生産活動モデルを表現する．
  - オブジェクト指向の継承と同じく，初めに基本的な入出力や情報の定義を行い，その定義を継承して再定義することによって，設計生産活動モデルの構築を行う方法が有効と考えられる．
- 多様な入出力対象表現
  EXPRESS によるデータモデルの記述や STEP 物理ファイルなども情報入出力の対象とすることによって，STEP における製品モデルとのインターフェースを持つ．入出力の対象としては，次の項目が考えられる．

- EXPRESS 記述などのデータモデルの構造に関する情報の蓄積
- STEP 物理ファイルなどのデータの蓄積

ただし，STEP におけるデータモデルの考え方に基づいた専用の記述言語仕様を設けるのではなく，本論文における他の対象表現と同じく，情報入出力対象として表現する必要がある．

- グループ化

設計生産に関与する組織構造を表現するために，次の項目を表現する必要がある．

- 工作機械やロボットなどのグループ化の表現，この場合は特定の工場の設備の中だけでなく，複数の工場設備を含んだ中での，仮想的なグループ化も含む．
- 設計者グループの，製品構造による機械系，電装系などのグループ，その中での部品の階層に応じた階層構造の表現．
- 常に決まった組織形態を維持するのではなく，設計活動の進展に応じて組織構成が変更される状況の表現．
- さらに個人の設計者が複数の設計プロジェクトに参加する状況の表現．
- 組織内部での，設計データの開示及び承認活動の表現
- グループ内での設計データの参照，設計データの修正変更，設計グループ構造などに関する種々の権利の設定表現．

- 部品など一般的な製品の入出力としての表現
  - 設計生産において加工される実際の製品，工作機械で用いられる工具，活動に費やされる人的な資源の表現
  - それらを用いる活動状況の表現
  - それらの搬送活動や現在位置，搬送状況の表現
- 並行して実行される活動の表現
  - 活動 (各設計プロセス) は，階層化，構造化される必要がある．また活動相互の関係を表現する必要がある．

上で上げた問題点を克服するためには，第3章で提案した情報入出力と入出力対象を用いる基本モデルの範囲内でこれらの対象を表現しなくてはならない．

5.2 本論文で用いるモデルの全体構造

前節 5.1.4における調査で明らかになった課題に対して，本論文では以下のような基本構造を摘要することによって解決を計る．前節では，多様な入出力対象を表現するためにモデルとしての全体構造を確立する必要性が指摘された．本論文での全体構造には，次の項目に関する全体構造を定義する必要がある．

- 各モデル間の継承構造
- 入出力を行う際の静的な構造

- プロセスを起動する際の動的な構造

これらの構造に関して，以下では静的な継承構造と入出力構造のモデリングに関しての提案を行う．

5.2.1 モデル間の継承構造：部品化によるリソース構造

複雑なモデル表現では，全体を平坦な構造で定義してしまうと，一部の変更が全体に影響を与えてしまうので，全体としての信頼性及び拡張性に欠ける恐れがある．従ってあらかじめ特定の目的と仕様を満たすようにモジュールを定義し，それを組み合わせることでモデル全体の信頼性を上げる必要がある．STEP では，幾何などの基本的なモデルを組み合わせることで，特定の分野で用いられる設計生産活動モデルを構築する．組み合わせられる基本モデルをリソースモデル，構築された製品モデルをアプリケーションプロトコルと呼んでいる [ISO95a] [鈴木 93]．

本論文においても，基本的な入出力の対象及び情報入出力の過程の表現をあらかじめモデル化し組み合わせる手法を用いてモデルを構築する．次に本論文では，設計活動者及び CAD/CAM システムなどの事物間の情報の入出力を，情報を直接交換するのではなく，必ずプロセスを経由する形で表現する．これは，複数の事物が関連する活動において，個別の情報交換の状況を集めただけでは，モデルの複雑化が予想されるためである．例えば，ある事物から見た複数の相手への入出力は表現できるので，1 対 1 又は 1 対 n の関係では問題ない．しかし n 対 n 関係の場合，情報入出力の挙動が全体としてどのような関係になっているかを表現することは難しい．そこで，すべての事物からの情報の入出力をプロセスのみに対して行うことにし，複数の事物の協調，並列挙動，及び活動の時間的関係などは，プロセス及び各事物間の情報入出力の時間的な関係として表現する．

本論文では，作業者，CAD/CAM システム，製品モデルのデータなどの基本的な入出力対象，及びその入出力対象に対するプロセスへの入出力仕様を予め定義しておき，それらを継承することによってモデル化を行う．現状では以下のリソースを考慮している．

1. 外部環境：設計者，CAD/CAM システム，工作機械など現実に存在するものを入出力対象とするリソース

2. データ：情報の集積を入出力対象とする抽象的なリソース

3. ディクショナリ：データリソース内のモデル構造に関する情報の集積を入出力対象とした抽象的なリソース

4. プロセス：入出力対象間の情報交換の表現

これらのリソース間の関係を図 27に示す． 2のデータリソースは，一般的なファイルシステムやデータベースの抽象化ととらえることができる． 3のディクショナリリソースは，2のデータリソースに蓄えられるデータモデルの構造に関する情報を蓄える．

実際のリソースのモデルでは，入出力されるメッセージの互換性を取るために特定のインターフェースと，そのインターフェースとの間で決められたメッセージの交換によってサービスを提供するプロセスの組み合わせでモデル化を行う必要があると考えられる．

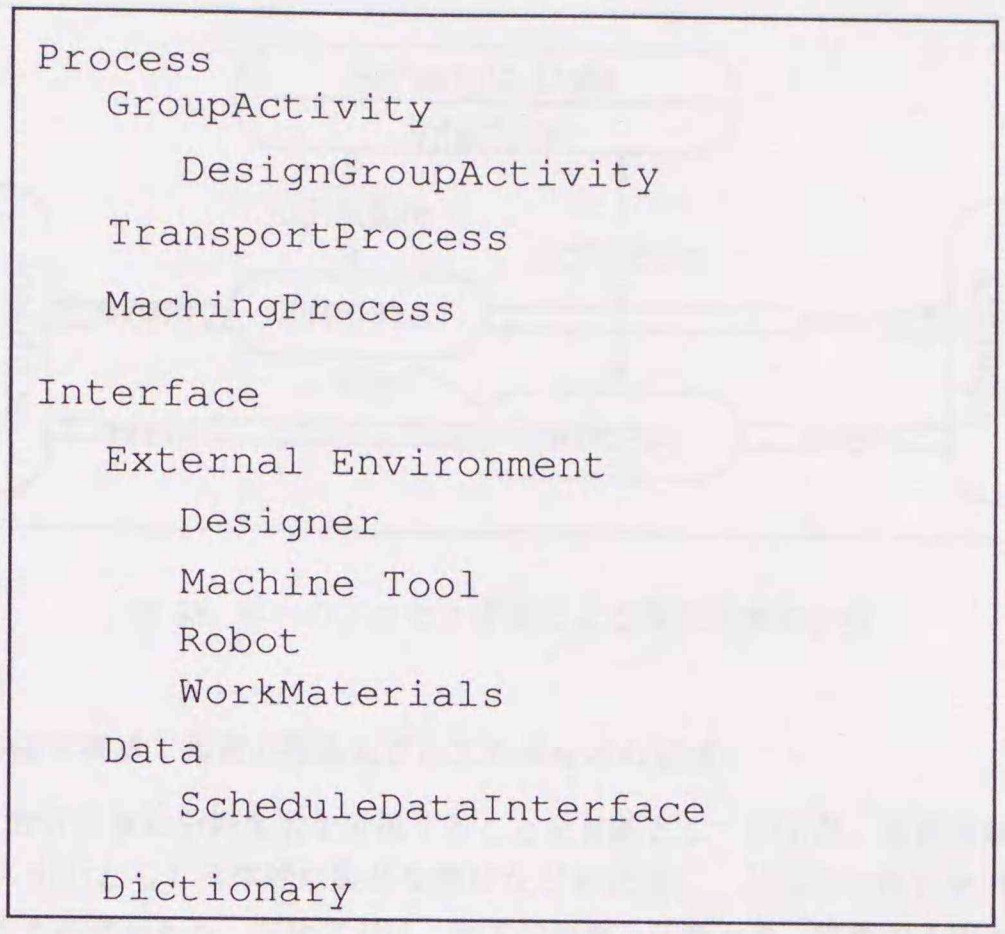


図 27: 入出力対象をリソースとして分類したモデル化

5.2.2 プロセスを起動する際の動的な構造：複数プロセスの並行動作の表現

第 2章で述べたように，複数の活動が同時に並行して行われる場合，別々の対象からの入力が前後して発生する可能性がある．例えば，複数の活動が行われるモデルでは，各作業者と所属するグループが別個に行動し，かつ相互に情報を交換する必要がある．このような状況では，各作業者やグループ間の情報の動きを逐次的な処理として表現することは難しい．そこで，プロセスや入出力対象に情報の入力が行われるごとに，これをイベントとし，対する反応としてプロセスの動作を定義するイベントドリブンシステムの形でモデル化を行っている．

次に，特定の活動に関するプロセスは基本的には 1 種類の定義で表現する．この定義からプロセスを必要に応じて再帰的に起動することで，稼動するプロセスを増やしていくこととする．また，プロセスの定義中で，イベントに対する対応と活動スケジュールなどの逐次的な情報を分離した形で定義する．これらの情報は，プロセスの外部にデータベースとして貯えられることとし，入出力用のインターフェースを定義する．各プロセスは，このインタフェースを通じてスケジュール情報にアクセスし，得られたスケジュール情報に従って，必要に応じて子プロセスを起動する．この構造を図 28 に示す．

プロセスを起動させるスケジュールの内容を変更することで，並行して活動するプロセスの，前後や親子の関係，終了の依存性などプロセス相互の関係を表現することが可能になる．

結果的に，特定の活動に関するプロセス及びインターフェースの定義は，その活動ごとに対象とプロセスの間でやり取りされるメッセージと，そのメッセージの到来をイベントしてのプロセスの反応の集合だけとなり明確で理解しやすい構造となる．

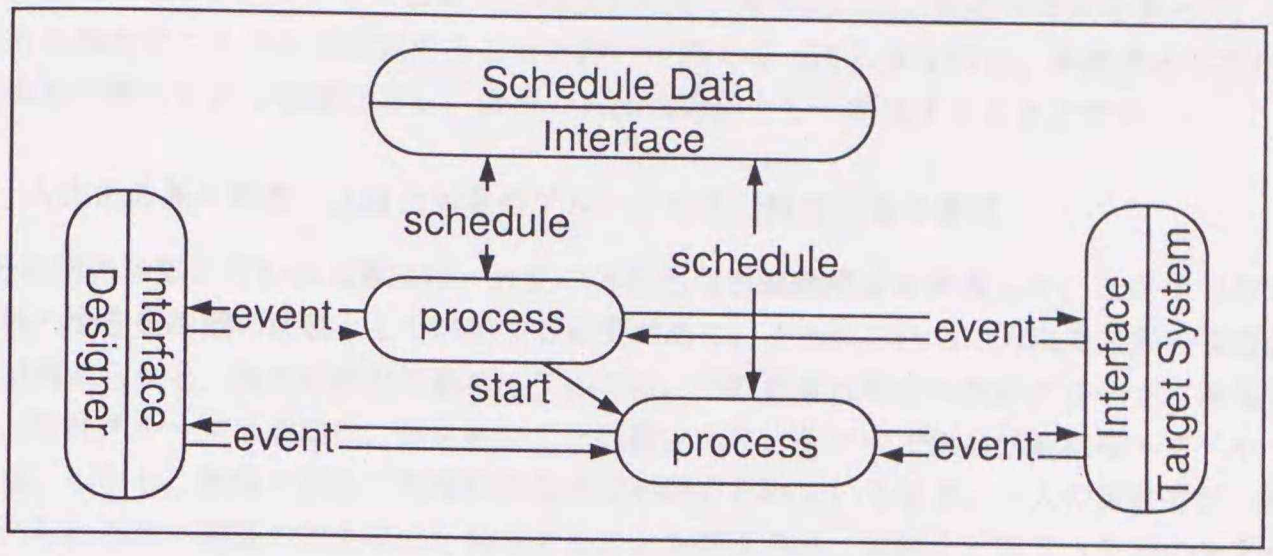


図 28: 単一のプロセス定義による複数活動の表現

5.2.3 入出力の基本構造：物資の搬送及び加工作業などの表現

本論文では，設計生産の分野全般を表現することを目的としているが，生産活動の表現では，加工プロセスへの入出力として具体的な製品や素材などを表現し，さらにそれらを用いた加工作業や搬送作業を表現する必要がある．本論文では，全ての対象と活動とを，情報の入出力の対象と情報の入出力現象として表現することを，基本モデルとして提案している．

この概念に従った物資の搬送や機械加工などの表現は，搬送される物資を単なる情報の単位 (オブジェクト) と考えるか，又は入出力対象と考えるかの違いに応じて，次の 2 つの方法に分けて考えることができる．

- 物資を情報単位としてのみ扱う
  移動される物資や被加工物を内部情報を保持した情報単位 (オブジェクト) として表現し，各対象の間の入出力として物理的な移動を表現する．搬送プロセスは，特定の機械装置から異動先の機械装置に対して情報単位を入力するプロセスとして表現される．この場合搬送される物体がどこに存在するか，あるいは搬送の状況がどのようになっているかの情報は，搬送プロセスの側で管理することになる．この表現形式では加工プロセスは，内部状態を保持した情報単位が入力された生産設備に対して加工情報を入力するプロセスとしてモデル化されることになる．[BB93] は，この形式に従って搬送システムによって運ばれる物資を情報単位として扱い，システム全体の挙動を LOTOS を用いて表現している．

- 物資を情報入出力対象として扱う
  被加工物や移動物体を内部状態を保持した入出力対象として，この対象への操作として加工プロセスをモデル化する方法が考えられる．この表現形式での物資の移動は，移動設備を通じて移動対象物に対して移動情報を入力するプロセスとしてモデル化される．この場合，搬送される物資がどこに存在するかどうかの位置情報は，入出力対象である搬送物資と搬送装置の間で管理されることになる．この表現形式での加工プロセスは，工作機械などの製造設備を通じて加工対象へ情報の入力を行うプロセスとして表現される．

前者と後者の違いは，対象となる物体を直接の入出力の対象として扱うか，別の入出力対象に管理される単なる情報単位として扱うか，の点にある．前者のモデルでは，情報単位 (オブジェクト)

を取り扱うための，プロトコルを設定する必要がある．結果的には，後者の方式を取って，素材などに対する入出力プロトコルを設定することと違いが無くなってしまうので，本論文としては後者の，被加工物及び搬送される物資なども，情報の入出力対象として表現することとする．

5.2.4 入出力の基本構造：入出力対象のグループ化及び階層構造の表現

設計生産の活動が行われる際には，表現の対象となる組織構造を表現して，その中でやりとりされる情報の動きを組織の活動として表現する必要があり，さらにこれらの入出力対象の構造が変化することが考えられる．例えば設計活動のモデルでは，各設計者は特定の設計グループに参加して活動を行う．設計グループは通常は，製品単位ごとに設けられ，その中で部品構成に従ってグループ訳が行われる．しかし，複数の製品で同種の部品構成が用いられている場合，一人の設計者が，複数の製品で用いられる同一種類の部品構成を担当することも考えられ，単純なピラミッド型のモデルを構成するだけでは，設計活動のグループモデルを構築することはできないと考えられる．さらに設計活動中では，各個人の設計者やグループ単位の間での情報入出力が行われるが，このグループ構成は，設計活動に進展に応じて変化することが考えられる．

このようなグループのモデル表現のために，本論文では活動しているグループごとにプロセスを定義することによって表現する．グループプロセスは，そのグループに含まれる対象，及び内部グループなどの情報を持つ必要がある．特定のグループに対する入力は，まずグループに対応するプロセスに対してなされ，そのプロセスがグループに含まれる対象に対して情報を分配する形で行われる．

グループの階層化は，このグループプロセスを階層的に関係させることで表現する．すなわち，特定の設計グループの中がさらに複数のグループで構成されている場合は，上位のグループへの入力が下位のグループに対してさらに分配されるようにする．

もし，グループ内での情報の入出力に関する権利を定義する必要がある場合は，活動プロセスが，適当なデータベースを参照することによって，情報の入出力を制限することによって表現できる．

新たなグループが入出力対象としてが組み入れられるような状況は，他の活動プロセスのと同じく，その生成や消去によって表現する．

この方式によって実際に存在する入出力対象としては，あくまでも個別の設計者や管理者あるいは生産設備が対象となり，活動中のグループはその時点において活動している仮装的なプロセスとしてのみ表現されることになる．

5.3 設計生産活動統合モデルの構築

本節では，前節 5.2で提案したモデルの基本構造を用いて，設計生産活動モデルとして設計活動と生産活動に関した活動モデルの構築を行う．

5.3.1 設計生産活動統合モデルの全体構造

ここでは，前節 5.2で提案したモデルの基本構造と，設計生産活動モデルとして構築される 設計グループ活動モデルと生産活動モデルとの関係に関して述べる．

設計グループ活動モデル及び生産活動モデルなどの設計生産活動モデルは，その活動のためのプロセスモデルと幾つかの入出力仕様のモデルとの組み合せとして表現される．プロセスモデル及びそれぞれの入出力仕様のモデルは， 5.2.1で述べたプロセス，外部環境，データリソースの下位型として定義される．

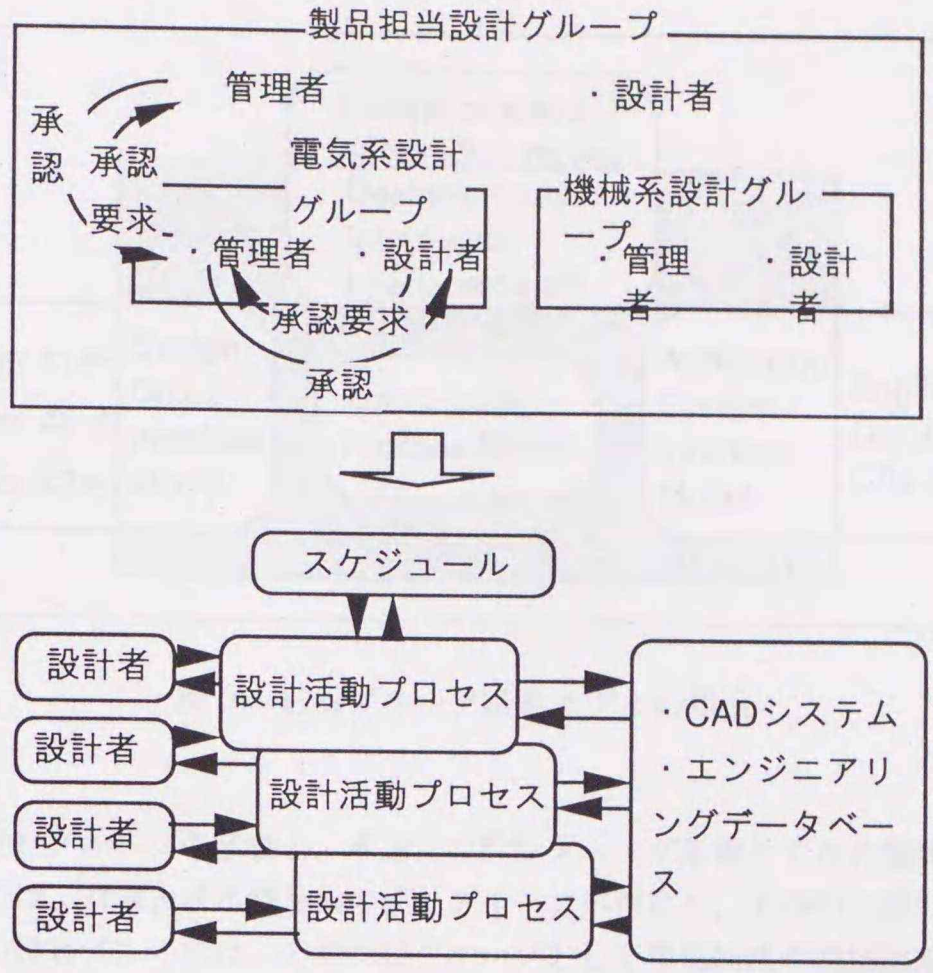


図 29: 設計グループの活動内容と設計グループ活動モデルとの対応

基本的な構造は，各入出力対象の間に，その時点で実行されている活動に応じた単数ないしは複数のプロセスが存在し，このプロセスがスケジュールデータと，グループ構造に設定された情報とに基づいて，各設計活動に対応した子プロセスを起動する．

それぞれの入出力仕様は，基本的には入力されるメッセージとその際に渡される引数によって定義されるだけで，入力を受け付ける相手に関する規定は行われていないので，別のモデルの中でも用いることが可能である．従ってモデルの中心となるのは，特定の活動向けのプロセスの定義である．別な言い方では，入出力する対象を限定して，その中での入出力に関する規定が具体的なプロセスモデルということになる．

上記の考えから 設計グループ活動モデルと生産活動モデルとは，それぞれの活動目的のために，設計者のグループや生産設備などの入出力対象やスケジュールデータなどと情報の交換を行うプロセスとしてモデル化される．別々の目的の活動モデルでモデル化が行われた場合は，その目的ごとのプロセスが起動されるので，同じ入出力対象に対する入出力が行われることはあっても，それぞれのプロセスの間で相互の情報交換が行われることは無いと考えられる．この複数のプロセスモデルの協調に関しては，5.4.3において考察を行うこととする．

他の分野における設計生産活動モデルの場合には，基本的には上の２つのモデルと同じ構造に従って拡張を行う．

5.3.2 設計グループ活動モデル

設計グループ活動モデルの構造

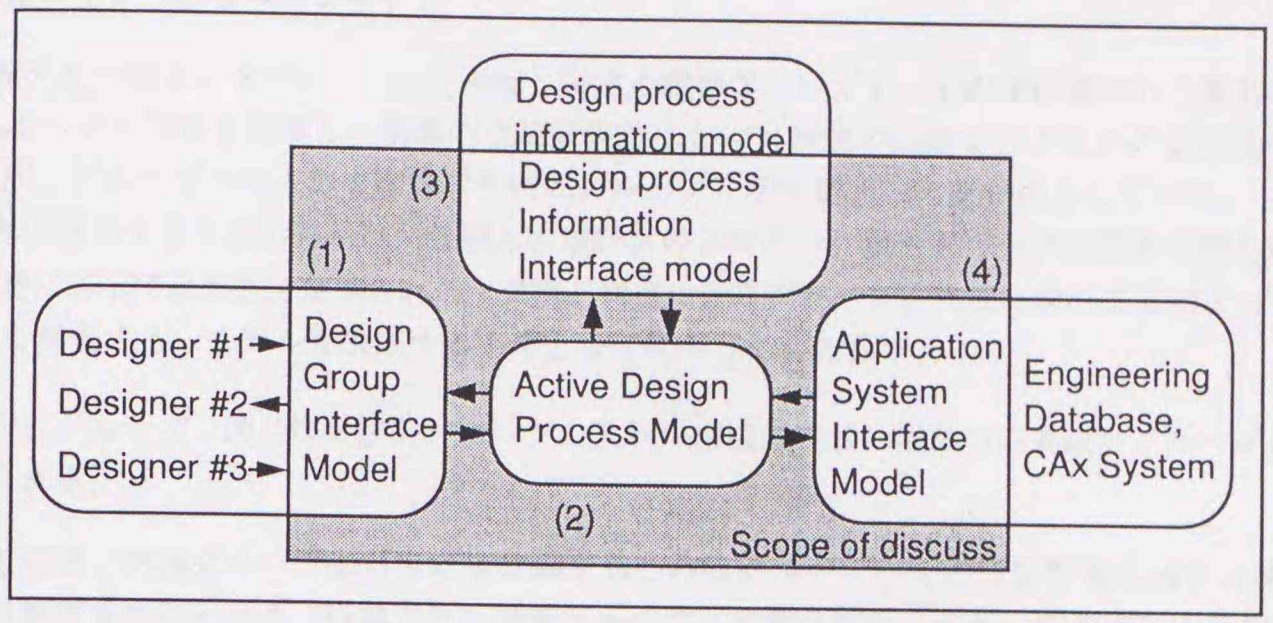


図 30: 設計グループ活動モデルの構造

図 29に実際の設計グループの活動と，本論文の設計グループ活動モデルの関係を示す．実際の設計グループの活動では，対象とする製品毎の設計グループが存在し，その中に管理者と作業担当者が存在する．製品毎の設計グループは，その内部グループとして電気回路や機械部品などの機能毎の設計グループを内胞し，その中にさらに管理者と設計担当者が存在する階層的な構造になっている．その中で各設計作業者は，自分が担当となっている設計部品データの仕様決定 (具体的な作図を含めた) 作業を行い，当該作業が完了した後に，担当グループ，その上位のグループによって承認を受ける．

この活動を本論文では，設計グループインタフェースモデルとアプリケーションシステムインタフェースモデルに対する情報の入出力の手順として設計過程を表現する．このモデルの中での設計プロセスが起動されることは，その設計グループのメンバに対して設計情報を入出力するためのソフトウェアのウィンドウが開かれると捉えることができる．設計グループのメンバは，そのウィンドウ（プロセス）の管理下で設計情報にアクセスする．

本論文では設計グループ活動モデルを，以下のプロセス及び入出力対象によって定義する．(図 30の (1)〜(4))

設計グループインタフェースモデル 各設計者，設計活動グループの階層構造を入出力対象として表現する．(図 30の (1))

動的設計プロセスモデル 設計者や設計グループによって起こされる設計活動の開始や中断，特定の設計データに対する参照，承認などの設計活動における出来事をイベントとして，プロセスがどのような対応を取るかを表わす．(図 30の (2))

設計プロセス情報モデル 各設計者や設計グループが担当する設計データ，その設計データに対する参照や変更に関する権利，管理者が変更できる各権利やデータなどの特権に関する情報，プロセスの時間的なスケジュール情報やプロセス相互の関係を規定する．(図 30の (3))

アプリケーションシステムインタフェースモデル CAD/CAM システム，エンジニアリングデータベースなどへのインタフェースを定義する．(図 30の (4))

設計グループインタフェースモデル

設計グループインタフェースモデルは，特定の設計グループと，その中に含まれる設計者個人や内部グループの関係を定義し，階層的な組織構造における情報の伝達を行うための仮想的な階層を作り上げ，グループへの入力を設計者や内部グループに割り振ることを目的としている．

実際に存在する入出力対象は，各個人の設計者のみであり，設計グループは設計活動の進行に伴い，必要に応じて仮想的に定義される．又同じ構造の設計グループを内部に含むことによって階層的な構造を構築する．モデルを構成する要素として次の３つの対象がある．

- 設計グループ：同じ様な設計グループの入れ子構造管理者，複数の内部設計グループから構成される．
- 管理者：内部グループに対する管理権をもつ内部グループ内での設計情報に対する情報参照，情報変更権利の変更，所属された設計グループに定義された，参照，変更権利を行使する．
- 内部グループ：その設計グループに含まれる下位のグループ．最終的には，管理者一人だけで，内部グループが存在しないレベルまで細分化される．これは，全ての階層を同じ構造にすることによって階層構造が単純化できることを目的として行っている．

一つの設計グループは，通常その内部に複数の管理者と複数の内部グループを持ち，その設計グループが対象とする設計データと，そのデータを取り扱うアプリケーションシステムが定義されている．ただしデータの変更権利は，その設計グループにおける管理者と，実際の作業者にしか与えられないと考えられる．

ある設計グループに含まれる内部グループ及び管理者は，その設計グループにおけるデータの参照権を引き継ぐ．ただし，これらの権利の構造は一般的なものであり，各設計グループが担当する設計データやアプリケーション，各管理者が持てる特権などに関する情報は，設計プロセス情報モデルの中で任意に定義できるものとする．この構造にマッチしないような情報に対する参照や変更の権利は，新たにグループを定義することで対処する．

このインターフェースモデルは，仮想的な階層構造を作り出すためだけに用いられ，特定のグループ活動における情報の流れは， 動的設計プロセスモデルにおいて表現される．又グループの構造，グループにおける管理者，そのグループに含まれる内部グループなどの構造，及び次に述べる設計プロセス情報モデルに蓄積される． 設計プロセス情報モデルには，グループの構造に関する情報以外にも，設計活動を実施するプロセスに関するスケジュール的な情報も蓄積される．

図31に 設計プロセス情報モデルに蓄積される設計データの参照，変更権利などのデータの例を示す．この図では，階層構造における権利の例として，電気回路設計における構造を示している．設計グループは，電気回路設計グループ (Electronics Design Group) と，その内部に含まれる ASIC(Apprecation Specific Integrated Circuit) 設計グループとから構成される．参照権利と変更権利とを行使するのは，そのグループの管理者である．設計データの参照権利は，基本的には上位グループから下位の設計グループが継承するので回路全体の情報を参照可能となる．しかし，変更権利は上位から下位に降りるに従って狭くなる形で継承される．具体的には回路設計グループ (の管理者) は，回路全体に対する変更権利を持っているが，ASIC 設計グループは，担当する ASIC の設計情報のみの変更権利を持つ．

図 32に設計グループインタフェースモデルで表現する仮想的なグループの構造を示し，図に沿ってモデルについて述べる． 設計グループインタフェースモデルは，一般的な設計組織を表現するためのモデルであり，上に示した設計グループと内部グループ，管理者などによって階層構造を表現す

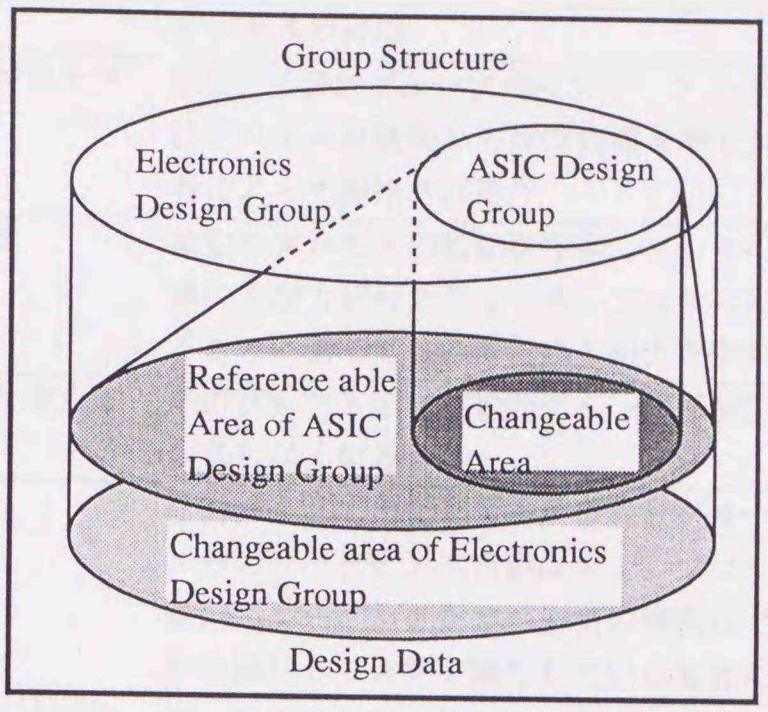


図 31: 階層化された設計グループにおける権利の構造

る．個別の設計者は，自分が属している活動のグループインターフェースを通じて，自分が参加している活動に対応するプロセスと入出力を行う．特定のグループへの入力は，このインターフェースを通じて，下位のグループへと配分される．より上位のグループに対して，例えば設計データを公開する場合は，作業者が対応する設計プロセスに対して所定のイベントを送り入力を受け取ったプロセスが予め規定されたリアクションとして，必要なグループに承認の要求を配分する．

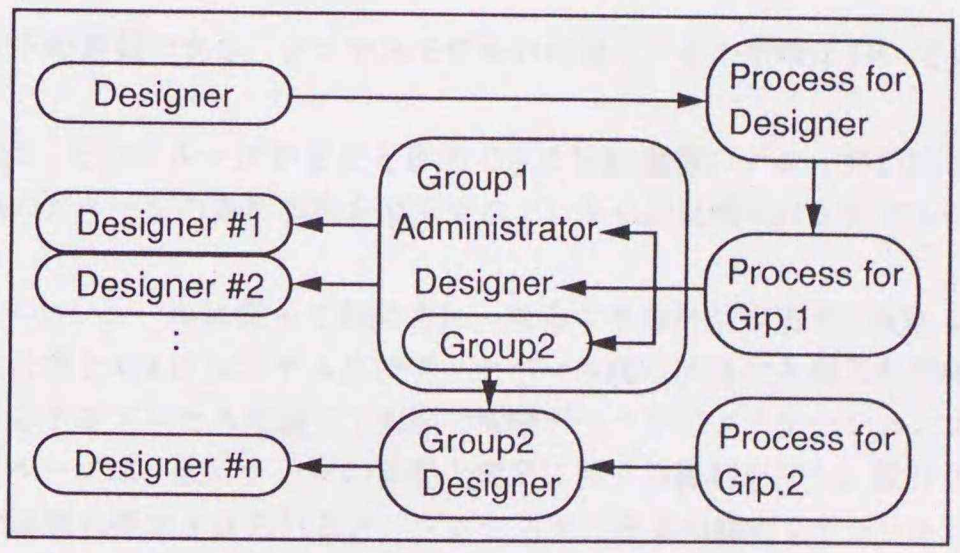


図 32: 仮想的設計グループの階層の実現

動的設計プロセスモデル

動的設計プロセスモデルは，設計グループが活動を行う際に，その活動中に関するモデルを定義する．プロセスの定義とは，実質そのプロセスに対して発生し，そのプロセスの内容や情報の伝達を

イベント	プロセスの対応
親プロセスからの子プロセス開始	該当する設計グループのインターフェースモデルが構築され，設計プロセスを経由して設計情報をやり取りする設計活動が担当者によって開始される．
プロセスの中断／再開	特定のプロセスの活動を中断して，そのプロセスを経由して情報の入出力が行えなくする．プロセスは再開命令以外を拒否することによって他の情報の入出力を中断する．
子プロセスの強制的な停止と消滅	そのプロセスの当該設計グループが解散され，活動そのものが存在しなくなる．
情報の入出力	設計プロセス情報モデル及び設計グループインタフェースモデルへのデータの入力活動などで，プロセスは，対象としている設計グループかまた参照変更の権利などを満たしているかどうかを確認して条件を満たしている場合に，情報の伝達を行う．
設計データの承認要求／承認	プロセスは，予め規定された対象に対して承認の要求を送る．
子プロセスの設計活動の終了	先に起動しておいた子プロセスが終了条件を満たして終了したイベント．プロセスは，必要に応じて新たな子プロセスの起動などを行う．
当該設計プロセスの終了	外部からの入力はないがプロセス内部で終了条件を満たしているかの判断を一定時間ごとに行い，条件を満たした時点で自身に対するイベントとして終了する．

表 9: 動的設計プロセスモデルに発生するイベントとプロセス内部の反応

変更させるイベントの定義である．プロセスモデルの内容も，その活動において発生するイベントのモデル化となる．

設計グループは，そのグループが変更と参照できる設計情報のグループが定義される．内部設計グループは，上位のグループの参照権利を継承する．しかし変更権利は上位グループの部分集合を継承している．

プロセスは，スケジュールに従って起動され，対応する設計活動者やCADシステムに関する情報を受け取り，設計者とCADシステム及びデータベース間の情報の入出力を制御する．設計者が活動を行う際は，対応するプロセスを通じて実際の活動データやアプリケーションプログラムにアクセスする．各設計グループは，設計データの参照と変更に関する権利を持ち，設計グループの作業者からデータの変更や参照の要求が送られると，プロセスが作業者の権利をチェックし，認められる場合に設計データを設計グループに対して公開する．

本論文では，設計活動のシナリオを用意し，その中で行われる入出力を分類整理し，設計者又は他のプロセスから設計プロセスに対して送られるメッセージをイベントとしてモデル化した．最終的には，用意したシナリオは次のイベントの組み合せで表現することができた．表9に各イベントとプロセスの対応を示す．

設計プロセス情報モデル

設計プロセス情報モデルは，設計プロセスの活動に必要とされるスケジュールなどの情報を定義するためのモデルである．本論文のモデルでは，必要なデータの項目だけでなく，そのデータを貯えるデータベースへのインタフェースとしても定義する．本論文では，上で述べたプロセスが活動を行うため，以下のデータを必要とした．

設計グループの階層構造と管理者 各設計グループに含まれる管理者及び内部設計グループの定義．

設計データグループ定義 設計グループの作業対象として扱われ，起動されるプロセスが入出力する設計情報を集めた設計データのグループ．

参照／変更権利定義 設計グループが，各設計データグループに対して認められる参照／変更権利／承認を行う必要があるかどうかの情報．

設計プロセス定義 設計プロセスの識別番号と対応する設計グループ，設計データグループに用いられるアプリケーションプログラムなどの付帯的な情報．

設計プロセス同士の関係 各設計プロセスが実行される際の，プロセスの親子関係，実行前に終了しているべきプロセスに対する依存性，平行実行の可能性などに関する情報．

アプリケーションシステムインタフェースモデル

アプリケーションシステムインタフェースモデルは，設計生産活動で用いられるアプリケーションシステムを表現するために用いる．外部から見た入出力特性と提供されるサービスとして定義する．

設計グループ活動モデルの動作例

設計活動中に発生する様々な出来事に関して，各設計者や設計グループ間で，どのようなメッセージが入出力されるかどうかを分析し，特定の入力イベントと，それに対する反応の集合としてモデル化できることを確認した．ここでは，その例として設計データの承認における活動の表現を図33を用いて述べる．図33では，縦軸方向が時間の進展を表し，横軸方向の矢印で各プロセス間の情報の交換を表現する．この図では，下に行く程時間的には後に起こった出来事となる．また，記述例を図34に示す．

設計データの承認活動 図33 Case 1 の動作例は，設計データの承認活動に関する情報の入出力を示している．設計データの承認では，作業者はグループインターフェースを通じて承認要求のメッセージを自分に対応している設計プロセスに対して発信する．

基本的には，承認を要求した設計者と，その設計者が属する設計グループのメンバー及び管理者の承認を得る必要があると考えられる．場合によっては，より上位の設計グループの管理者に承認を得る必要がある可能性もある．従って，その設計データではどの設計グループの管理者によって承認を得るべきかを規定するデータが決められている必要がある．一般には，データの変更の権利又は義務を追っている者の間で承認を得る必要があると考えられる (この場合儀礼的な責任ではなく，実務上の責任に関してである)．各プロセスの挙動は，次のようなものである．

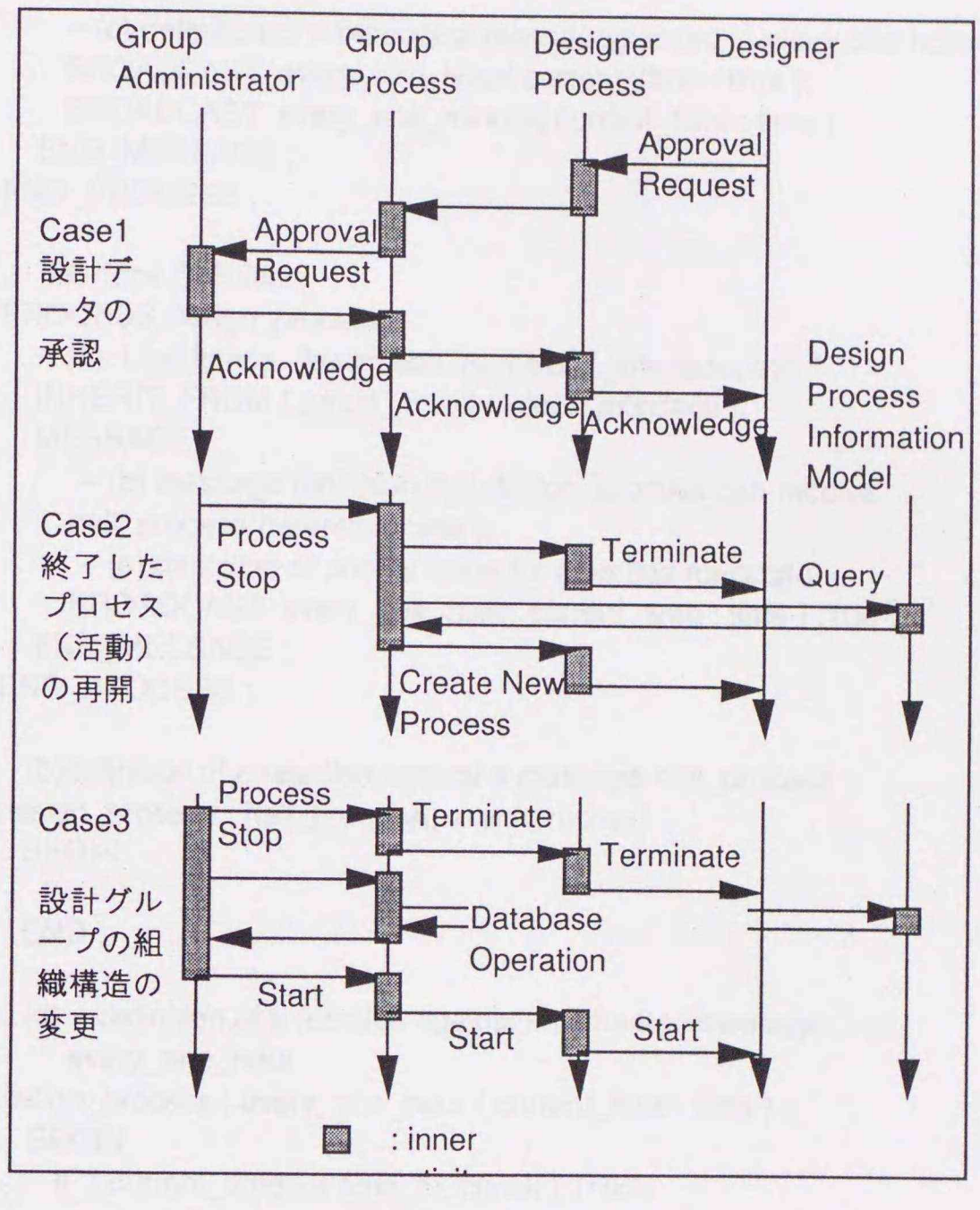


図 33: 設計グループ活動モデルの動作例

```
PROCESS clock_interface ;
   MESSAGE
     -- (c) definition of a broadcast message passed in every one hour
     BROADCAST  every_one_hour( current_time : time );
     BROADCAST  every_one_minute( current_time : time );
   END_MESSAGE ;
END_PROCESS ;

 -- Interface Definition
PROCESS design_process ;
    -- (a)  inheritance  description from clock_interface, etc.
    INHERIT_FROM ( group_activity, clock_interface );
    MESSAGE
        -- (b) message definition that design_process can receive
       halt_process(parent:process );
        -- (e) definition of priority value for inherited message
       BROADCAST  every_one_hour( current_time : time ) :100 ;
    END_MESSAGE ;
END_PROCESS ;

-- (b)definition of a reaction against a message halt_process
design_process : halt_process(parent:process) ;
  BEGIN
        ....
  END ;

-- (d) redefinition of a reaction against an inherited message
--      every_one_hour
design_process : every_one_hour ( current_time : time ) ;
   BEGIN
     IF ( current_time == time_to_check ) THEN
     ; -- redefined action  for design process
END ;
```

図 34: 設計グループ活動モデルの記述例

1. 承認要求のメッセージを受信したプロセスは，設計者が属する設計グループ及び予め規定された承認を必要とするグループのプロセスに対して承認要求を送信する．
2. 受信した設計グループのプロセスは，設計グループの管理者に対して承認要求を送る．
3. 結果を承認要求を初めに出したプロセスに返す．
4. 承認要求を受け取って配信を行った設計プロセスは集計した結果を設計者に返す．

プロセスの再開 (データの修正) 図 33 Case 2 では，設計グループの管理者は，単純にデータの承認あるいは非承認のメッセージを返す (図 33 Case 1) だけでなく，場合によっては当該設計プロセスを中止して，既に完了した設計プロセスを再開させてデータを修正させる．

設計グループの組織構造変更 図 33 Case 3 ではさらに，設計プロセス情報モデルの中の設計グループの構造やスケジュールに関する内容を変更することによって，産み出される 設計グループインタフェースモデルの構造を変更して，設計活動を行う組織の構造そのものを変更することができる．

むろん Case 2 や Case 3 におけるプロセスの操作や組織構造の変更は，それが許される特権を持っている管理者によって行われる必要がある．

以上で述べた例から，本論文におけるモデルでは一般的な設計活動や承認作用のみならず，設計活動の中断と既に完了した活動の再開，仮想的な設計グループの構成変更までを表現できることを示した．

5.3.3 生産活動モデル

生産活動モデルの概念

生産活動モデルは，生産現場において行われる加工や搬送などの作業を，情報の入出力の観点から表現することを目的としている．実際の作業と生産活動モデルの関係を図 35に示す．即ち，現実には工作機械や搬送設備によって行われている加工や搬送作業を，それぞれの目的のプロセスがデータベースからスケジュールと必要な情報を取り出して，対応する機械設備に対して入力する形で表現を行う．

本論文では生産活動モデルを，以下のプロセス及び入出力対象によって定義する．(図 36の (1)〜(4))

生産システムインタフェースモデル 生産現場における各種の生産設備，及び生産設備のグループ階層構造を入出力対象として表現する．(図 36の (1))

搬送生産システムインタフェースモデル 生産現場における各種の搬送設備，及び搬送設備のグループ階層構造を入出力対象として表現する．(図 36の (2))

動的生産プロセスモデル 生産設備及び搬送設備などのグループによって起こされる加工／搬送作業の開始や中断，特定の製造データに対する参照，加工／搬送作業における出来事をイベントとして，プロセスがどのような対応を取るかを表わす．(図 36の (3))

生産プロセス情報モデル 各搬送設備や生産設備グループに入力される加工データ，プロセスの時間的なスケジュール情報やプロセス相互の関係を規定する．(図 36の (4))

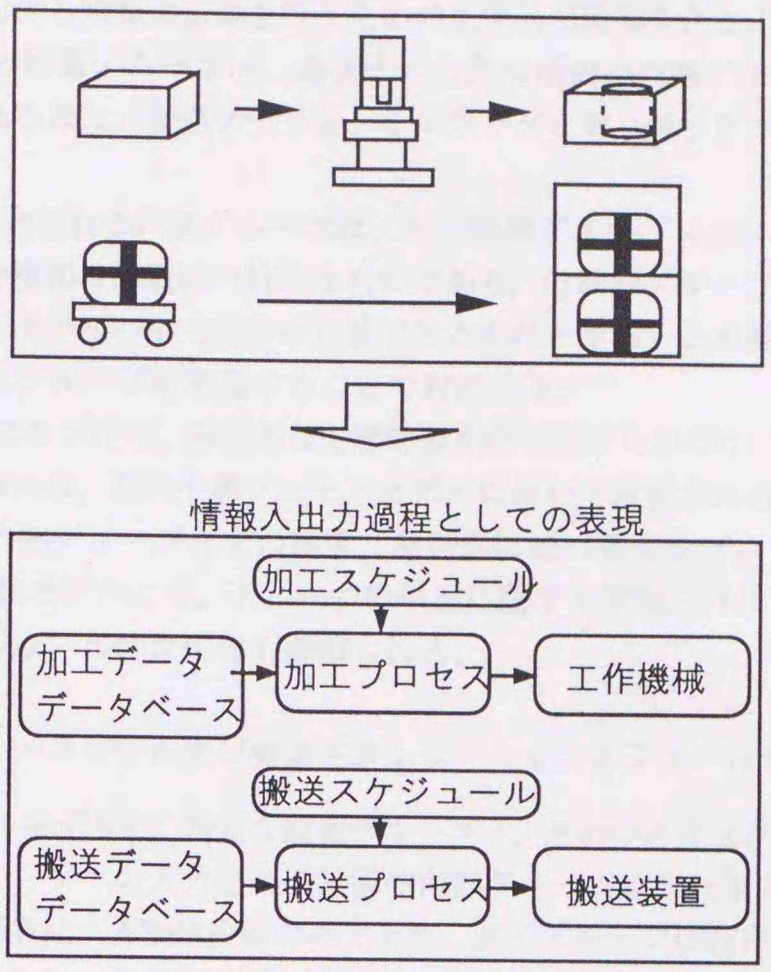


図 35: 生産活動モデルの概要

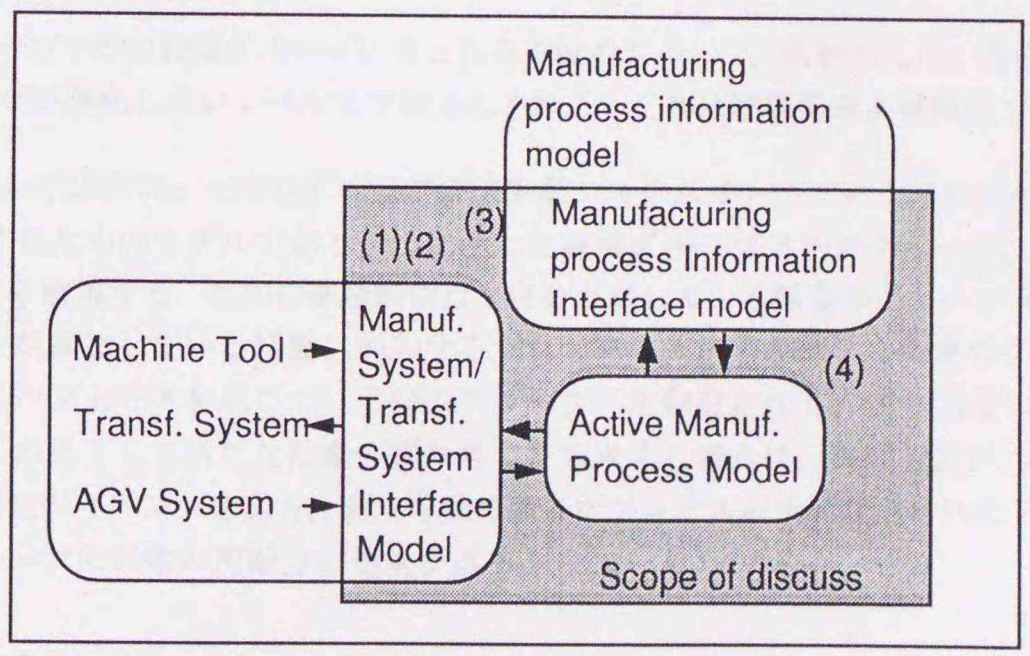


図 36: 生産活動モデルの構造

生産システムインタフェースモデル及び搬送生産システムインタフェースモデルは，生産設備や搬送設備のグループにおける情報の伝達を行うための仮想的な階層を作り上げるためのインタフェースモデルである．一つの設備グループは，通常その内部に複数の内部グループを持ち，その設備グループに対して入力される加工／搬送データと，そのデータを取り扱うアプリケーションシステムが定義されている．

ある設備グループに含まれる内部グループは，その設備グループにおけるデータの参照権を引き継ぐ．ただし，これらの権利の構造は一般的なものであり，各設備グループが担当する製造データ情報は，生産プロセス情報モデルの中で任意に定義できるものとする．この構造にマッチしないような情報に対しては，新たにグループを定義することで対処する．

このインターフェースモデルは，仮想的な階層構造を作り出すためだけに用いられ，特定のグループ活動における情報の流れは，動的生産プロセスモデルにおいて表現される．又グループの構造，そのグループに含まれる内部グループなどの構造，及び次に述べる生産プロセス情報モデルに蓄積される． 生産プロセス情報モデルには，グループの構造に関する情報以外にも，生産活動を実施するプロセスに関するスケジュール的な情報も蓄積される．

生産システムインタフェースモデル及び搬送生産システムインタフェースモデル

このインターフェースモデルは，特定の設備グループと，その中に含まれる製造設備や内部グループの関係を定義し，グループへの入力を製造設備や内部グループに割り振ることを目的としている．実際に存在する入出力対象は，各製造設備のみであり，設備グループは設計活動の進行に伴い，必要に応じて仮想的に定義される．又同じ構造の設備グループを内部に含むことによって階層的な構造を構築する．

- 設備グループ：同じ様な設備グループの入れ子構造，製造設備及び複数の内部設計グループから構成される．
- 製造設備：所属された設備グループに定義された加工データに従って活動を遂行する製造設備．
- 内部グループ：その設備グループに含まれる下位のグループ．最終的には，設備一つだけで，内部グループが存在しないレベルまで細分化される．これは階層構造が単純化できるためである．

生産システムインタフェースモデル及び搬送生産システムインタフェースモデルは，一般的な製造設備を表現するためのモデルであり，上に示した設備グループと内部グループ，製造設備などによって階層構造を表現する．個別の製造設備は，自分が属している活動のグループインターフェースを通じて，自分が参加している活動に対応するプロセスと入出力を行う．特定のグループへの入力は，このインターフェースを通じて，下位のグループへと配分される．より上位のグループに対して，例えば加工が終了して新たな活動の割り当てを要求する場合は，製造設備が対応する製造プロセスに対して所定のイベントを送り，要求を受け取ったプロセスが予め規定されたリアクションとして，必要なグループに活動の再配分の要求を送る．

動的生産プロセスモデル

このモデルでは，搬送プロセスと加工プロセスは，並列に別個に実行されるプロセスではなく，特定の時間的な関係の下で協調して実行されるプロセスとして表現される必要がある．そこで動的

生産プロセスモデルとして，それぞれの製造設備に対して情報を出力するプロセスとしてモデル化を行う．

設備グループは，そのグループが変更と参照できる設計情報のグループが定義される．内部設備グループは，上位のグループの参照権利を継承する．

プロセスは，スケジュールに従って起動され，対応する加工設備や搬送設備に関する情報を受け取り，加工設備及び搬送設備とデータベース間の情報の入出力を制御する．各設備が活動を行う際は，対応するプロセスを通じて実際の加工データや搬送データが送られる．各設備グループは，加工／搬送データの参照に関する権利を持ち，設備グループの製造設備からデータの参照の要求が送られると，プロセスが製造データを設備グループに対して公開する．

本論文では，基本的には動的設計プロセスモデル と同じ構造で次のイベントの組み合せで表現することができた．

1. 親プロセスによる製造プロセス開始 (該当する設備グループの構築及び加工／搬送活動の開始)
2. プロセスの中断／再開
3. 親プロセスから強制的に停止される場合のプロセスの消滅．(当該設備グループの解散)
4. 加工／搬送データの入力など生産システムインタフェースモデル，搬送生産システムインタフェースモデル及び，生産システムインタフェースモデルへの情報の入出力
5. 子プロセスの活動の終了
6. 当該設計プロセスの終了 (外部から入力されないが，プロセス内部で終了条件を満たしているかの判断を一定時間ごとに行い条件を満たした時点で，自身に対するイベントとして終了する)．

生産プロセス情報モデル

生産プロセス情報モデルは，設計プロセスの活動に必要とされるスケジュールなどの情報を定義するためのモデルである．本論文のモデルでは，必要なデータの項目だけでなく，そのデータを貯えるデータベースへのインタフェースとしても定義する．本論文では，上で述べたプロセスが活動を行うために，以下のデータを必要とした．

設備グループの階層構造と製造設備 各設備グループに含まれる管理者及び内部設備グループの定義．

製造データグループ定義 設備グループの作業対象として扱れ，起動されるプロセスが入出力する加工／搬送情報を集めたデータグループ．

参照／変更権利定義 設備グループが，各製造データグループに対して認められる参照権利の情報．

加工／搬送プロセス定義 プロセスの識別番号と対応する設備グループ，製造データグループに用いられるアプリケーションプログラムなどの付帯的な情報．

製造プロセス同士の関係 各加工／搬送プロセスが実行される際の，プロセスの親子関係，実行前に終了しているべきプロセスに対する依存性，平行実行の可能性などに関する情報．

生産活動モデルの動作例

図 37に，生産活動モデルの動作例を示す．

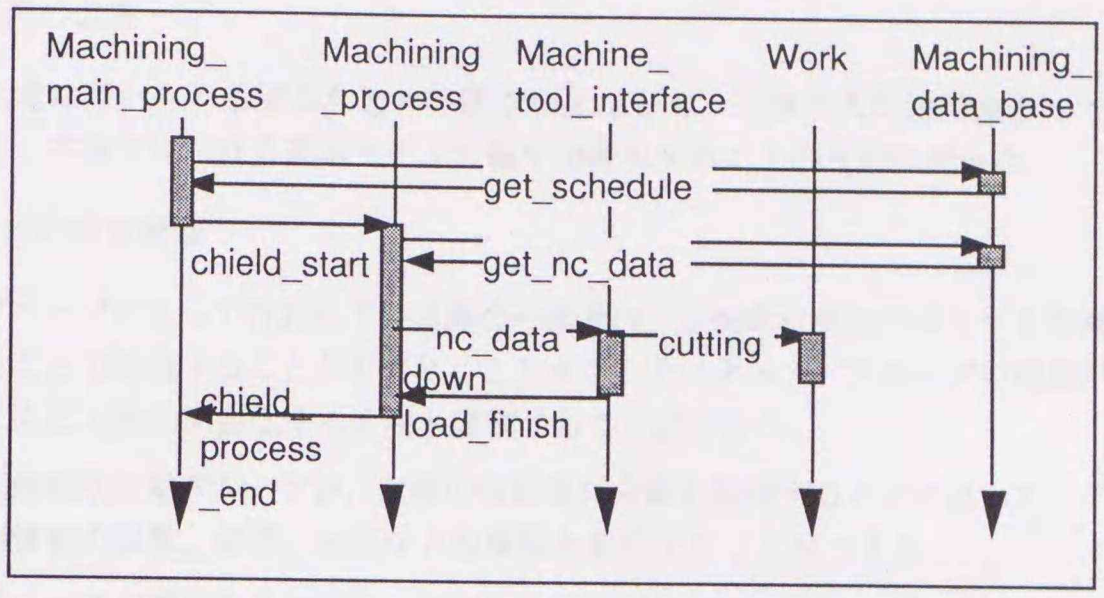


図 37: 階層構造化された設備グループの動作例

5.4 特徴及び他のモデルとの比較

5.4.1 本論文におけるモデル構造の特徴

5.2で提案したモデルの構造では，次の項目を特徴としてあげることができる．

- リソース構造
  - 特定の目的に応じたリソースを定義することによって，設計生産活動モデルの構築を容易にすることができる．
  - 一般的なファイルシステムやデータベースの抽象化や，データモデルの構造に相当する入出力対象を定義することで，モデル構造の構築，モデルデータの入力などの活動もモデル化することができる．
  - 活動に関する全ての情報をプロセスを経由させることによって，複数の対象が関与する複雑な活動を，特定のプロセスを時系列的に通過する情報の流れとして表現できる．
  - 本論文ではモデリングにおいて，静的なモデリングの継承構造と，プロセスを中心とした入出力におけるリソース構造，特定分野のプロセスを起動する際の再帰的起動に関して，3 種類の基本構造を提案した．
- 並行プロセス表現
  - 特定の活動に関しては，基本的には一種類のプロセスをスケジュールに従って再帰的に起動させることによって，スケジュールデータと各活動ごとのプロトコル定義を分離することができた．
  - スケジュールとプロセスの定義を分離することによって，プロセスの並行動作の可能性や，活動の終了に関する依存性，などをアルゴリズムの定義とは別に，スケジュールデータの中で表現することができる．
  - 特定の活動におけるスケジュールとプロトコルを分離することによって，プロセスの表現を，活動ごとに入出力される情報と，その情報の入力をイベントとしての対応とに明確化することができた．

- 搬送状況の表現
  - 生産に用いられる部品などの移送や加工なども，情報の入出力対象として表現することで，本論文における基本モデルの範中で表現することが可能になった．
- グループの階層構造
  - グループによって行われている場合の活動は，仮想的な活動プロセスを再帰的に活動させることで表現することができた．こうすることによって，グループの構造が，活動の進展に応じて動的に変化する状況を表現することができる．
  - 仮想的な活動グループが，情報の交換及び分配を制御することによって，グループ組織内の情報の閲覧，操作，承認などの権限を表現することができる．
  - グループの構造変更に関する権利も上記と同等に定義することができる．
- 拡張性
  - 本論文で提案する設計生産の活動は，同じ基本構造に基づいて，それぞれの分野ごとのプロセスとインタフェースとを定義することでモデル化される．これらのモデルの各プロセスは，ともに並行して活動することが可能である．
  - 別個の目的のプロセスやインタフェース仕様でも，それらを継承させることで，容易に統合することができる．

5.4.2 他のモデルとの比較

5.1.3で調査した既存の研究と提案した本論文における一般的モデル構造とを，5.1.4で上げた問題点に関して比較する．

- 全体構造 (フレームワーク)
  - 5.1.3で調査したSTEPとIEMは，基本的には静的な継承構造に関するもので，入出力の基本構造やプロセス起動の基本構造などに関するモデルはあまり見られなかった．これに対して，本論文では，継承構造，プロセスの起動の構造，入出力の構造に関しての構造化を新たに提案することができた．
- 複数の部分モデルを組み合わせるための機能
  - 本論文で提案した設計生産活動統合モデルでは，設計活動，搬送活動及び加工活動などの複数の活動モデルの構築を行った．これは，STEPにおけるモデル構造に関する問題と同じであった．本論文では，同様な活動に関しては，それらの共通の上位型としての入出力モデルを定義する必要があった．
  - 本論文での継承モデリングは，オブジェクト指向におけるものと同じであり，継承を行う階層構造の構造化に関して大きな問題点があった．この問題点は，MMS，STEP-EXPRESSにおいても同様と言える．
- 多様な入出力対象表現

- 本論文で提案した入出力対象の表現には，データの蓄積やデータ構造に関する記述などがある．この方式を用いることで，例えばSTEPにおけるデータモデルから物理ファイルへの写像を表現することなども可能になった．

- グループ化

  - 本論文では，仮想的な階層構造を構築することによって，活動の進展に応じて組織構造が変化する状況にも対応できることを示した．これは，MMSやSTEP-EXPRESSにもない特色と言える．

- 部品など一般的な製品の入出力としての表現

  - 本論文では，全ての存在を情報の入出力対象として表現する形式を用いている．従って，生産設備，生産設備によって加工される被加工物，搬送設備，搬送される工具なども情報の入出力の対象として表現し，これらの対象間での情報の入出力によって，加工や搬送の状況を表現する．

5.4.3 複数プロセスの協調についての考察

本論文における活動モデルでは，複数の種類のプロセスが協調しあう状況を表現する必要があった．例えば，生産活動における搬送と加工の活動では，時間的な依存性があり，被加工物の搬送が終わって固定された後に加工を行い，加工が終わった後に再び被加工物の搬送を行う必要がある．

全く異なった領域の活動相互の関係を表現するためには，何等かの形での基本的な構造が必要と考えられる．

このような状況のモデル化では，それぞれ別個のプロセスとしてモデル化する方法が考えられるが，この形式では，異なるプロセス間での関係付けるために，プロセス相互で入出力されるメッセージの定義をする必要がある．通常のモデリングでは親子関係でプロセスの起動や停止などのメッセージをやり取りすることが行われるが，これ意外に関与するモデルのプロセス毎にプロセス相互に交換されるメッセージを定義する必要がある．しかし，この方式を単純に用いた場合は，プロセスのモデルの依存性が高くなってしまうことが懸念される．

他の方法としては，本論文で用いたような多重継承による方法が考えられる．即ち，実際に行われる設計活動や生産活動のそれぞれを予めモデル化し，必要な活動モデルのプロセスの定義から多重継承を行ってモデルを組み合わせる．これらのプロセスは，それぞれの目的のプロセスとして起動されて並行して入出力を行うが，データベース上での前後関係などの記述では，同じ種類のプロセスとして取り扱うことができる．

今後のモデル化の方針としては，関連させるプロセスから多重継承を行って，新たに，モデルを構築する方法を用いる必要があり，モデルの階層構造の確立が今後も問題となると考えられる．

5.5 結 論

第5章では，第3章で提案した基本モデル及び第4章で提案した設計生産プロセス記述言語に基づいて設計生産の活動を表現するため，対象と考えられるモデルに関する分析を行い，さらに既存の研究報告における問題点の抽出を行った．得られた問題点を解消するために設計生産活動を表現するための基本的なモデル構造である設計生産活動統合モデルを，その構造に基づいた設計グループ活動モデル及び生産活動モデルの提案を行い，以下の結論を得た．

- 本論文のモデリングの対象として考えられる設計グループ活動モデルや生産活動モデルに関する調査を行い，表現対象や表現される活動，モデルの構造に関して次の項目での問題点が明らかになった．
  - 設計者，設計者のグループ，グループの管理者，データを入出力するCADシステム，データを貯蔵するデータベースなどを設計活動で関与する対象として考える必要がある．
  - 設計者グループは一般的には製品の構造によって，機械系，電装系などに分けられ，その中では部品の階層構造に応じた階層構造を取っている．
  - 設計者グループは，常に決まった組織で運営されている訳ではなく，その組織構造は仮想的なものであり，設計活動の進展に応じて組織構成が変更されると考えられる．一つのプロジェクトに複数の作業者が関係することは無論のこと，個人の設計者が複数の設計プロジェクトに参加することも考えられる．
  - 生産活動に関与する対象として，ロボット及び工作機械などの生産設備，AGVやコンベアなどの輸送設備，実際に加工される被加工物や工具などを表現する必要がある．
  - 部品など一般的な製品の入出力として，設計生産において加工される実際の製品，工作機械で用いられる工具，活動に費やされる人的な資源の表現，
  - 多様な入出力対象表現として，EXPRESSによるデータモデルの記述やSTEP物理ファイルなども情報入出力の対象とすることによって，STEPにおける製品モデルとのインターフェースを持必要がある．
  - この対象による活動として，通常の設計データの入力や修正の活動，データが完成した後での公表，承認活動が考えられる．チームリーダーは，活動，活動のゴール，デッドラインなどに関する連絡を可能にし，チームメンバーによる活動の進展に密着し続け，個別の決定の効果をチーム全体に対する公表を可能にする必要がある．
  - 設計データの参照，設計データの修正変更，設計グループ構造などに関する種々の権利を設定する必要がある．
  - 活動(各設計プロセス)は，階層化，構造化される必要がある．また相互の関係を表現する必要がある．
  - 生産設備を用いる活動状況や搬送作業を表現する必要がある．
  - 一定のモデル構造を確定しておき，その構造に従って要素モデルを組み合わせてモデルの構築を行う手法を用いる必要がある．
  - 基本的な情報入出力の構造，対象，及び過程に関するモデルを予め用意しておき，その内容をオブジェクト指向と同じく継承して再定義することによって，個別の設計生産活動モデルを構築する方法が有効と考えられる．
  - 常に決まった組織形態を維持するのではなく，設計活動の進展に応じて仮想的な組織が構成され，その中で各種の活動が行われる状況の表現．
- 本論文で設計生産に関する活動を表現するための設計生産活動統合モデル の提案を行った．これは上に挙げた諸問題を克服して，設計生産活動における種々の対象と仮想的な組織を表現し，その対象間で実行される活動を表現するために，基本的な入出力対象とプロセスのモデル化を行い，それらを継承して設計生産活動モデルの構築を行うものである．

- 設計生産活動統合モデルの特徴は，特定の活動において起こるイベントを一種類のプロセスとしてモデル化し，スケジュールなどのデータに応じて個々の活動に相当するプロセスを起動して情報の入出力を制御する．階層的な構造を作る入出力対象のモデルを用いることで，仮想的な作業グループの構造を任意に作り出すことができる，などがある．

- 5.1.1及び5.1.3 節でまとめた課題に対して他のモデル化方法論との比較を行い，本論文で提案したモデル構造が有効であることを明らかにした．

- 全く異なった目的の活動が相互に関与する状況 (例えば搬送と加工) を表現する方法に関して考察を行った．一度，別々にモデル化を行い，組み合わせた活動として表現する場合には，多重継承を行って複数の活動に関するイベントを受け付けることを可能にし，スケジュールからは一種類のプロセスとして扱えるようにすることによるモデル化の方法を示した．

6　プロセス管理機能実装の検証

本章では，形式記述表現された設計生産活動を用いる計算機支援システムについて述べる．この分野では具体的な方法論が確立されていない．しかしその中でも計算機支援システムの機能としての並列に実行されるプロセスとその管理機能の実装は，並列作業シミュレータだけでなく，各種の設計生産の活動の統合と管理を行う際の基盤技術として重要と考えられる．従って形式表現された活動の取り扱いと並列プロセスの管理技術に重点をおいて次の項目に関する報告を行う．

1. 設計生産活動の形式表現を実装する一般的な形態を提案する．
2. 特に表現されたプロセスの実装を計算機を用いて行う場合のシステムの構成と必要な機能とを明確にする．
3. 上の機能を実現する計算機支援システムの基本的な構造を提案する．このシステム内部で実行されるプロセス間の通信，プロセスの生成及び消去，などを管理する機能をプロセス代数を用いて定式化する．
4. 定式化したプロセス管理機能を実装したシミュレータシステムの試作を行う．

本章は，6.1で設計生産の分野で形式表現されたモデルを計算機システム中で実装する報告に関して調査し，計算機支援システムに必要とされる要件を抽出する．この調査結果に従い，6.2では本論文におけるシステムの基本構造を提案する．また，3.2.2で定式化を行ったプロセスの実行管理機能の実装可能性を件称するために行ったプロセスシミュレータの開発について6.3で述べ，6.4及び6.5で総括及び考察を行い，結論とする．

6.1　設計生産プロセスの実装のための用件と分析

6.1では，設計生産に関する形式仕様記述言語を用いた表現の計算機による計算機支援システムの試作開発の問題点を確認し，解決するために既存の研究に対する調査を行い問題点の抽出を行う．

6.1.1　設計生産活動の形式表現を扱うための機能分析

ここでは形式表現された対象を扱う計算機支援システムに関して調査を行い，記述表現された設計生産活動の計算機による実装で必要とされる機能を明らかにする．調査の対象として次の研究報告を用いる．

- CIMOSA [Esp93] [Ver94] [Ver93]
- STEP [ISO95b] [木村 95] [小島 93] [菊地 92]
- LOTOS を生産活動表現に用いた報告 [BB93]

CIMOSA

CIMOSA の初期の資料 [Esp93] では，MSS 言語ではなく IDL (Implementation Description Language) と呼ばれる言語を用いて企業モデルを記述し，これによって記述されるモデルを IDM (Implementation Description Model) と呼んでいる．さらに MFW (Modeling Framework) と IIS (Integrating Infrastructure) の概念が用いられ，これらの 3 つが大きな課題となっている．

このIDLの計算機実装における特徴としては，形式仕様記述言語を用いた表現だけではなく，実際のプログラムに対するプログラミング言語と同じく，生産システムにおける実行言語としての位置付けが当初からなされている点がある．このためIDMの部分モデルは，サブルーチンパッケージと同じ扱いをされている．

また，実オペレーションのためにリリースされるIDMは，IEOE (Integrated Enterprise Operation Enbironment) において，統合化インフラストラクチャによって一般的な計算機言語へ変換したりコンパイルすることなしに実行可能となっている．この点が，CIMOSAと今日用いられている他のモデリングテクニック又はソフトウェア開発テクニックとの相違点である．しかし全体として閉じたモデルを構成しているため，相互の依存性が高くなり他の環境などとの相性が悪くなることが考えられる．

MFWで得られるモデル，IDLで記述されたモデル，及びIISサービスによる実行のための形態の間には，現在のところIISの中でサービスファンクションとして実行されるIDMの中での構築を行う幾つかの変換 (コンパイラ／インタプリタ) が必要とされている．

図38にIDLによる記述からIISへの流れを示す．ここでのIEEEは，Integrated Engineering Enterprise Environmentを示す．IDMは，IISのサービス関数として実行される．IDLはIISで認識

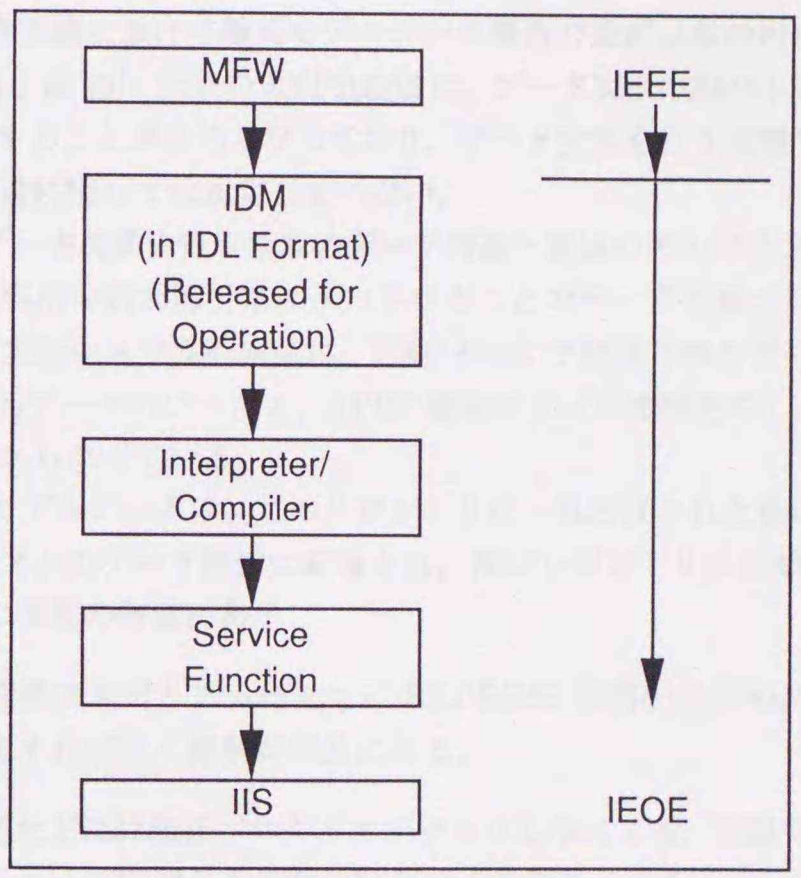


図 38: CIMOSAにおけるIDLからIDSへの流れ

されるサービス関数のレベルよりは概念的である．また幾つかのノード上での物理的な分割は，ユーザに対しては不可視でなくてはならない．

MFWのIISサービス関数への接続は，インタープリタによる高レベルのIDL構成要素からIISの低レベルのサービス関数への翻訳となる．この翻訳は，IEOEの中で行われてIISサービス関数を呼び出す形で行われるか，外部で別のベンダーによって行われてIISサービス関数を外部から呼び出す形で行われるかの違いがある．

外部のベンダーによって翻訳が行われる場合は，IDL 記述から IIS サービス関数の呼び出しの方法が複数存在する可能性がある．従ってこれらの解釈に関する規定が必要となる．また IEOE の中で解釈を行う場合は，IDL 記述から IIS サービス関数への翻訳方式を CIMOSA 内部で規定する必要がある．

形式仕様記述言語を用いて表現された生産システムの実装に関して，CIMOSA における問題点は，形式記述表現から予め用意されている ISS サービス関数への翻訳 (写像) をどのように規定するかであった．本論文では，実装形態を予め規定する必要はないが，CIMOSA における問題は記述表現された内容を特定の計算機環境で実行させる際にも同様に発生すると考えられる．

STEP

STEP における形式仕様記述言語 EXPRESS [ISO95b] [木村 95] の機能は 1.2において述べた．この STEP 及び EXPRESS に関する派生的な研究として，EXPRESS で記述されている STEP モデルと他のアプリケーションのモデル間のマッピングを定義するための EXPRESS-M [Bai] 及び，STEP の ARM (Application Reference Model) と AIM (Application Interpreted Model) 間の写像を表現するための EXPRESS-V [Wen] などが提案されているが，現状としては試作のレベルにあり，また直接計算機環境での実装を目的としてはいない．

EXPRESS をデータ交換における製品モデルデータ構造の記述以外の用途で用いた例として，小島らによるものがある [小島 93]．元来の EXPRESS は，データ交換の際の中間形式である STEP データのモデル構造を表現することが目的となっており，データ交換を行う対象である他のシステムにおけるデータモデルの構造に関しては規定していない．

小島らの報告は，データ交換を行う対象のデータ構造と変換のアルゴリズムとを EXPRESS を用いて記述し，これらを専用の構文解析系に入力させることでデータ変換プログラムを自動生成させるものである．データ変換システムの構成は，EXPRESS で記述されたデータモデルを貯蔵しておくリポジトリと呼ばれるデータのプールと，STEP 物理ファイルや特定の CAD システムのデータ構造への入出力ルーチンからなっている．

入力された STEP モデルデータは，このリポジトリに一旦貯蔵された後に，一つ一つ取り出されて変換する相手のシステムのデータ構造に変換され，再びレポジトリに格納される．

この方式では，次の項目の特徴がある．

- データの記述と変換アルゴリズムにともに EXPRESS を用いるため，EXPRESS で記述された内容だけに注視すれば良く理解が容易になる．
- リポジトリの操作などには直接プログラマがタッチしなくても，EXPRESS におけるデータ構造と変換アルゴリズムの記述から自動的に生成することができる．
- 変換する相手のデータを EXPRESS における Derived Attribute (誘導属性) とすることで直接の変換アルゴリズムを呼び出す部分が無くてもデータ変換を行うルーチンを指定することができる．

同様に EXPRESS をデータ交換におけるアルゴリズム記述に用いたものとして，[菊地 92] によるものがある．この方式では，データ変換を行う際の元のデータモデルの構造と，変換後のデータモデルの構造の両方を EXPRSS を用いて記述し，この EXPRESS による記述と入力されるデータの両方を一つのファイルにまとめた形で用意し，データ交換を行うごとに，データ変換用プログラムを

自動的に生成するものであった．しかしこの方式が提案された時点では，STEP 物理ファイルに関する規格が確定されていなかったため，小島等の研究のようにリポジトリを用いていない．

そのために，入力されるファイル構造に対する対応が不完全であったり，データ変換を行うアルゴリズムの起動を陽に指定する必要があった．

これまで調査した [ISO95b] [木村 95] [小島 93] [菊地 92] の各報告には以下の特徴がある．

- データ構造などの記述の対象である言語と実装されて実際の変換を行う言語とは，言語変換プログラムを用いて別個の言語を用いている．
- 記述された内容を計算機内部で蓄積して，必要に応じて記述された内容を変換する方式を用いている．
- 実際に変換を行うシステムの中の挙動を直接にプログラムすることなしに形式仕様記述言語による記述のみで必要なシステムの生成を行うことができるようにしている．

また実装形態への写像について次の項目の問題点があった．

- 計算機システム上での EXPRESS 表現されたデータ構造とアルゴリズムの実現方法
- そのデータ構造に対する別の CAD システムファイルからの入出力
- 計算機システム上でのデータに対する変換アルゴリズムの摘要

本論文でも上であげた特徴を用い，又課題となっている実装形態への写像に関する問題を克服することを念頭において，活動に関する形式仕様記述言語による記述表現の計算機システムでの実装を計る必要があると考えられる．

LOTOS

二木 [二木 92] からは形式仕様記述言語を用いるメリットとして，次の 2 点が指摘されていた．

- 仕様自体を計算機システムによって処理することによって，記述の妥当性を機械的に判定できる可能性がある．
- 仕様の内容に基づいた計算機支援システムを自動的に実装することの可能性がある．

LOTOS を用いる通信システムの仕様記述では，この方法を用いて仕様の妥当性の判定及び通信システムそのものの実装に用いている．このメリットは，形式仕様記述言語を用いている全ての方式で得ることができ，本論文においてもそのメリットを期待することができる．

Biemans らによる研究 [BB93] では，生産システムにおけるシステムの挙動を LOTOS を用いて記述し，システムの挙動の検定を行っている．この報告の中では，LOTOS を用いた形式表現を行った際のメリットとして ESPRIT プロジェクト 410 'SEDOS' (Software Environment for the Design of Distributed System) の中で行っているシンタクスチェッカ，シミュレータ，テスター，ベリファイア，コンパイラが開発中であり，これらのシステムをそのまま流用できる利点が指摘している．

LOTOS を用いたこの研究報告では，形式仕様記述言語を計算機で実装する最も典型的な例であり，本論文においても同等の効果が期待できる．

6.1.2 計算機支援システムに必要な機能のまとめ

これまでに行った調査から，形式仕様記述言語による記述を実装するシステムにおける特徴を次の２つの視点からまとめる．

- 記述表現の妥当性に対する判定結果や，記述内容の別形態への変換結果などの出力形式．
- 表現されたプロセスを実際にシステムの内部で実行させる際の実行形式．

以下では，初めにシステムから出力されるデータの種類に関して整理し，次に表現されたプロセスの実行形式に関してまとめる．

計算機支援システムからの出力の種類

設計生産活動に関する形式表現を入力とするシステムから出力されるデータの形式は，次の種類に大別することができる [菊地 95b]．

1. 記述そのものの出力：
   設計生産に関する形式記述を情報として保存し，システムからの出力とする形式である．
2. 形式記述の内容妥当性判定や他の形式への変換：
   形式記述を読み込み，判定又は変換処理の結果を出力する形式である．出力される情報として次の項目がある．
   (a) 記述の妥当性の判定を行った結果：
   仕様を形式仕様記述言語を用いて表現した場合の利点として，記述の妥当性の機械的な判定の可能性が上げられている [二木 92]．また各研究報告においても，形式表現の利点として上げている [ISO95b] [Esp93] [BB93]．
   妥当性の判定は，構文上の妥当性，及び意味上の妥当性について行う．形式言語の構文仕様を形式的に表現し，コンパイラコンパイラに入力することによって，構文解析系を生成する．この構文解析系に記述表現を入力することで構文上の妥当性に関しての判定を行う．意味上の妥当性の判定は，構文上で妥当であると判断した後に行う．形式表現では，表現する元となる基本的な言語要素のもつ意味が明確に限定されているので，その組み合わせによって表現される記述の無矛盾性，完全性，非曖昧性などの意味上の判定を機械的に判断する．
   (b) 他の形式に変換した結果：
   形式仕様記述言語の仕様に対応した図式表記方法がある場合，この図式表記への変換出力が上げられる．また逆に図式表記から言語形式への変換も考えられる [ISO95b]．
3. 実行形式への変換出力：
   形式記述表現された活動モデルを実行可能な形式に変換して出力する．実際には，特定の対象向けにデータを出力するプロセスを，次の実行形式に変換する．
   (a) アプリケーション向け入力形式への変換: 工作機械用 CL データ，ロボット言語，CAD/CAM システムの操作マクロなどへの変換が考えられる．

(b) プログラム形式への変換：データ構造や操作を定義するクラスライブラリ，SQL などのデータベース定義言語への変換が考えられる [Esp93] [木村 95]．データモデルの形式仕様記述言語 EXPRESS をアルゴリズム記述として，データ交換システムを開発する方式も報告されている [木村 95]．

(c) 専用システムの入力として，そのままの形式で処理系に入力する．

実際の変換は，特定の対象向けにデータを出力するプロセスを定義し，これをそれぞれの実行形式の書式に合わせて変換して出力することで行う．単純にデータを出力するだけの形式であれば，問題は少ないと考えられるが，条件判定やくり返しなどのアルゴリズムの変換は，出力する対象向けデータの構造との対応を計る必要がある．

実行形態の分類

次に，表現されたプロセスを具体的に実行させる場合の，実行形態についてまとめる．実行形態は，1) 表現されたプロセスを別のアプリケーションへの入力として変換して実行させる形式と，2) アプリケーション内部に読み込んで直接実行させる形式の二つに大別することができる．

1. アプリケーションへの入力としての変換／実行：
   形式仕様記述言語で表現された活動を，既存のアプリケーションや生産設備などに入力できる形式に変換して実行させる．出力は，前項の 3 の (a) であげた各形式が考えられる．データモデル用の形式仕様記述言語 EXPRESS [ISO95b] をアルゴリズム記述とし，データ交換システムを開発する例もある [木村 95]．

   このシステムでは，変換されるプロセスが単純にデータを出力する形式の場合は，変換は容易にできると考えられるが，条件判定やくり返しなどは，出力する対象向けデータの構造との対応を計る必要がある．この形式では，アプリケーション内部で基本的には単一のプロセスしか表現することができない．従って同時に並行したプロセスを実行させることは難しいと考えられる．

2. プログラムとしての実行：
   形式表現されたプロセスを，プログラムとしてアプリーション内部に組み込んで実行させる．プロセスを実行させる形態として次の項目が考えられる．

   (a) 表現されたプロセスをシミュレーションシステムの下で実行させ，同時に並行して実行される各グループプロセスの挙動を確認する [Esp93]．

   (b) 形式表現されたプロセスを実行プログラムの形式にして，OS 又はアプリケーションプログラムから，部品として実行させる．

   (c) 形式表現されたプロセスを実装する専用のシステム環境を開発し，記述表現されたプロセスを直接実行可能な形式に変換して実行させる．所謂 PDM システムのような設計生産活動の制御や CIMOSA における実行は，この形式に分類することができる．

通常は記述されたプロセスを逐次処理を行うプログラムに変換するだけでは，同時に並行して行われるプロセスを実現することができないと考えられる．そのためには，並行プロセスを実行可能なオペレーションシステムの下でプロセスの生成や消滅，プロセス相互の交信を実現する必要がある．

又 CIMOSA の実装に関する問題と同じく，設計生産プロセス記述言語で記述した活動をシステム内部で実行させるためには，そのシステム内部で実際に実行される機能と設計生産プロセス記述言語で記述されている内容の写像関係を規定する必要がある．

これらの調査から，記述表現された設計生産活動の計算機による実装で必要な幾つかの機能が判明した．その中で形式記述表現の読み込み，蓄積，特定の形式への変換部などは，その形式ごとの対応で解決できるので重要とは考えられない．プロセスの実行では，既存のシステムの入力形式に変換を行うだけでは，単一のプロセス実行しか実現できず，本論文の特色である並列活動プロセスの記述表現と計算機による実装を実現することが難しい．従って次章以降の課題として，並列プロセスの実行と管理技術を実現する必要があると考えられる．

6.1.3 本論文における課題

前節 6.1.1での調査と 6.1.2における分類に基き，設計生産活動の記述表現を用いる計算機支援システムので必要とされる機能に関してまとめる．プロセスに関する記述を逐次処理を行うプログラムに変換するだけでは，同時並行に行われるプロセスを実現することができない．そのためには，プロセスの生成や消滅，プロセス相互の交信を，並行プロセスを実行可能なオペレーションシステムの下で，計算機支援システムが実行する必要がある．計算機支援システムの機能として，次の点に関して考慮する必要がある．

1. 4.2.1で定式化されたプロセス動作の管理機能を実現する．
2. 単なるシミュレーションとしてシステム内部で並行プロセスを実現するだけでなく，種々の実装 (インプリメンテーション) が行えるように考慮する．例えば，具体的なアプリケーションに依存しない，クラスライブラリなどの形に，必要に応じて変換して実装できる方式で開発を行う．
3. 並行して実行されるプロセスを実現するためには，直接計算機言語で記述することも可能であるが，設計生産活動の表現の目的にあった言語の方が理解し易い．そこで，活動表現は専用の形式仕様記述言語を用い，その活動を計算機環境内で実現する場合には計算機言語に変換して用いる方式が望ましい．ただし，設計生産プロセス記述言語から実際にプロセスを実行するプログラム体系への変換コンパイラを開発する必要がある．
4. 設計生産プロセス記述言語による記述から具体的な実装への写像について対応を取る．
   - 設計生産プロセス記述言語で記述表現された入出力インターフェースとアルゴリズムの計算機システム上での実現
   - そのデータ構造に対する具体的な外部入出力対象からの入出力の実現
   - 計算機システム上でのプロセスの起動の指定方式
5. シンタクスチェッカ，シミュレータ，テスター，ベリファイア，コンパイラなどの開発への方向付けを行う．

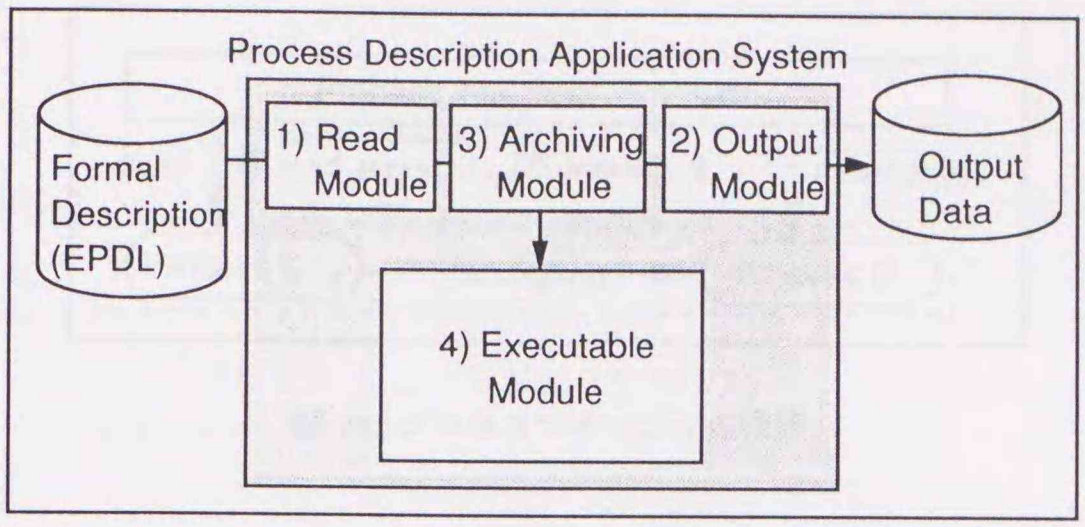


図 39: 本論文で提案する計算機支援システムの一般形

6.2 計算機支援システムの提案

設計生産活動の形式記述表現を用いる計算機支援システムの一般形を提案し，前章での調査結果で重要であったプロセスの実行管理部の定式化を行う．プロセスを実行させる計算機支援システム内では，設計生産プロセス記述言語 [菊地 95a] によって記述されたプロセスを具体的に実行させる必要がある．しかし，情報の伝達やプロセスの生成などを管理する機能は，記述言語の仕様では陽には定義されていない．又これらの機能を形式仕様記述言語で表現することもできないため，この形式仕様記述言語が基本としているプロセス代数を用いてプロセス実行管理機能の定式化を行う．

6.2.1 計算機支援システムの基本構造

これまで行ってきた計算機支援システムに関する機能の調査に基づいて，設計生産の形式記述表現を実装するシステムの一般形を提案し，その構造を図 39に示す．

システムは，モデュールの組み合わせとして提案する．このように計算機支援システムの基本構造をシステムをモデュールの組み合わせとして提案することで，計算機支援システムに必要な機能を整理することができる．また，設計生産プロセス記述言語に基づいた PDM システムや生産支援システムもこの一般系に対してさらに設計グループや CAD システム，データベースなどへのインターフェースを付加した拡張，または入出力対象として生産現場で用いられる各種の工作機械やロボット搬送装置などを付加した拡張として考えることができる．

以下では各部の詳細を図 39の番号に沿って示す．

1. 入力部分 (図 39の (1) 部)
   設計生産プロセス記述言語の構文解析を行い，3 で述べる形式で保存できるように変換する部分．設計生産プロセス記述言語の構文上の間違いなどは，この部分で検知して結果を出力する．

2. プロセス表現の変換出力部分 (図 39の (2) 部)
   形式記述されたプロセスを，別のアプリケーションで実行できる形式に変換して出力する．

3. プロセス表現の保存部分 (図 39の (3) 部)
   形式表現されたプロセスを，計算機で実行可能な形式に変換し，メモリ上又はディスク上に保存する．

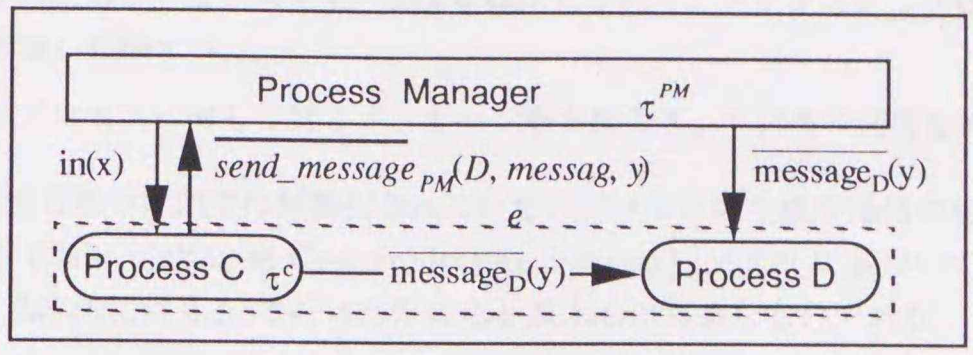


図 40: プロセスマネージャの機能

4. 実行管理部 (図 39の (4) 部)
   各プロセスの生成，消滅，及びプロセス同士や外部との通信の管理を行い，記述表現されたプロセスを実行させる．さらに具体的なシステムの場合は，この実行管理部の下で他の対象と入出力を行う部分が必要となる．

上で提案したシステムの構成では，(1) 及び (2) の部分は特定のフォーマットごとの変換で一般化した構造を提案できない．従って，(3) のシステム上での実行を前提としてプロセス記述を保存する構造と，(4) の並行プロセスをソフト的に実行させて管理する実行管理部の機能とが重要になる．

6.2.2 プロセス実行管理機能の定式化

第 4章で提案した形式仕様記述言語では，プロセスから他のプロセスへの入出力や子プロセスの生成や削除は，プロセス同士で直接行う形で表現していたが，計算機支援システムを実装してシステム内部に，設計生産プロセス記述言語で記述したプロセスが稼動している際に，この活動が計算機支援システム内部でどのような形で実現されるかについては規定していない．

またこの問題に関連して，メッセージの配送や新しいプロセスの起動を行う存在としてプロセスマネージャを用いて 4.2.1で提案した基本モデルの CCS による定式化を行い，実際に EPDL で記述表現したプロセスを実装する際には，プロセスマネージャで実行されている機能を計算機支援システム内部で実装する必要性を指摘されている．

これらの問題点に対して，本論文では次の方法によって解決をはかる．

1. 設計生産プロセス記述言語で記述されたプロセスやインターフェースは，そのままシステム内部で実現される形式を取る．

2. 計算機支援システムの内部でプロセスの通信や実行の管理機能を持つプロセスを導入し，プロセスマネージャと名付ける．

3. プロセスマネージャの機能は，4.2.1で提案されたプロセスマネージャの機能を実装する．

   図 40にプロセス C とプロセス D 間の入出力状況とプロセスマネージャとの関係を示す．EPDL による形式表現では，図の点線で囲った部分として入出力が表現されるが，実際にはプロセズマネージャを経由し，その管理の元で入出力が行われる．プロセスマネージャの主な機能は，次の 3 点が上げられる．

   - すべてのプロセスの生成や消滅を管理する．

- 総ての入出力をプロセスマネージャを経由して行うことによって，プロセス同士の通信状況を管理し記録する．
- 複数のプロセスに対して同じメッセージを分配して，1 対多の通信を実現する．

4. 形式仕様記述言語の仕様では詳細化されていないプロセスの生成や通信の管理機能をプロセス代数の CCS (Caluculation of Communicating Systems) [Mil89] に基づいて定式化する．この定式化は，基本的には 4.2.1で行った定式化と基本的に等価なものである．

   4.2.1で行った定式化を再び行うことは，紙面の都合上行わないが，計算機支援システム内部のプロセスの挙動をプロセスとプロセスマネージャの活動として定式化して表現できることを示した．プロセスマネージャに必要な機能をまとめると以下の項目になる．

   (a) プロセスの平行動作の実現
   (b) プロセスの生成，識別子の一意性を確保した上でのプロセス識別子データベースへの登録
   (c) 他のプロセスからの要求に対応するプロセスの消滅，プロセス識別子データベースからの削除
   (d) 終了したプロセスのプロセス識別子データベースからの削除
   (e) プロセス同士の通信の実現／並行活動しているプロセスとプロセスマネージャの通信の実現
   (f) 複数のプロセス集合を指定された際の同じメッセージの発信
   (g) プロセスのタイプを指定された際の同じメッセージの発信
   (h) 他のプロセスからの要求による，プロセスの活動状況の確認

6.3 プロセス管理機能実装の検証

6.3.1 シミュレーションシステムの目的

本論文では，並行プロセスの動作を実行管理する部分に重点を置いて計算機応用システムの開発を行う．また，この部分だけでは実行結果を表示することができないので，各プロセスの活動状況を表示する部分を付加してシミュレータとする．

その目的としては，以下の項目が上げられる．

- 4.2.1で定式化したプロセス管理機能の実現の可能性の証明
- 開発後の他の方面への実装の可能性を示す．

これは，本論文で提案した，基本モデルと形式仕様記述言語に基づく設計生産活動の計算機支援システム全体の開発を行う場合には，4.2.1で定式化した機能が実際に実装可能であり，プロセスの動作や相互の交信が正しく行われ，かつその交信の状況を活動の履歴として保存できることを示す必要があるためである．

システム全体としては他のアプリケーションなどとのインターフェースや，設計生産プロセス記述言語から開発用言語への変換用コンパイラの開発を行う必要があるが，それらの開発課題は上に上げた目的から見ると副次的なものであるので，本論文では複数プロセスを発生させて管理するプロセス管理機能の実装可能性の検証に傾注して行った．

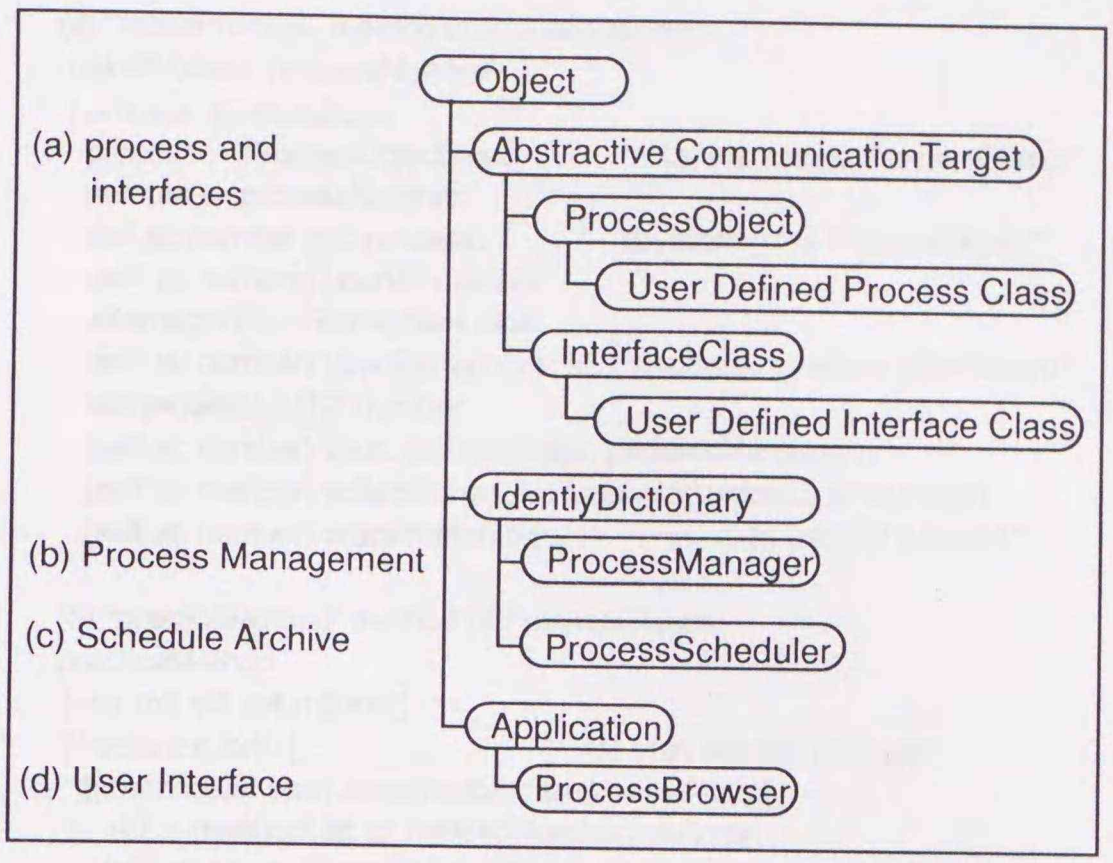


図 41: シミュレーションシステムのクラス階層構造

6.3.2 シミュレーションシステムのクラス階層構造

本論文で提案した形式仕様記述言語では，プロセスの特性の継承機能を用いていることから，シミュレーションシステムは，Macintosh 上で稼動するオブジェクト指向言語の Smalltalk/V2.0 を用いての試作を行った．シミュレータ全体は，次に示すクラスの集まりとして開発した．図 41 にクラス階層構造を示し，図中の番号に添って各クラスについて述べる．

1. ProcessObject(図 41 (a) 部)
   ProcessObject は，形式仕様記述言語で表現されたプロセスを，シミュレータ中でオブジェクトとして扱うためのクラスである．このクラスは，Smalltalk のクラス階層中の Object クラスのサブクラスとして定義する．作業プロセスなどのユーザ定義によるプロセスは，Process クラスのサブクラスに UserdefindProcessClass として定義する．

2. ProcessManager(図 41 (b) 部)
   ProcessManager は，前節で定式化したプロセスの生成，消滅，プロセス同士や他の対象との情報の通信などの管理を行う．ProcessObject 同士が行う情報の交換は，全て ProcessManager が仲介することによって行われる．このクラスは，IdentityDictionary のサブクラスとして定義する．

3. ProcessScheduler(図 41 (c) 部)
   このクラスは，プロセスの生成や相互の情報通信に関するスケジュールを貯えるために用いる．本来は，前節で提案したように形式表現された入出力対象の仕様に応じた構造で実現して，外部からスケジュールのデータを読み込んで蓄える構造にする必要があるが，本論文では，IdentityDictionary クラスのサブクラスとし，予め初期値を与えることとした．

```
(a) "makeProcess" method of ProcessManager
makeProcess: processNumber
 ¦ process aSemaphore¦
  process := ProcessObject new.          "-- (a) instantiate ProcessObject"
  number := processNumber.
  self at: number put: process.          "-- (b) storing the ProcessObject"
  (self at: number) yourID: number.
  aSemaphore:= Semaphore new.                "-- (c)"
  (self at: number) yourSemaphore: aSemaphore. "-- share aSemapore"
  self processList12: number.
  (self at: number) start: self message: processMessage .
  (self at: number) collectSchedule: (processSchedule at: number).
  (self at: number) practiceMethod!        "-- (d) start parallel process"

(b) "practiceMethod" method of ProcessObject
practiceMethod
 ¦ m1 m2 m3 m4 m5 m6 ¦
 Processor fork: [                        "-- (e) start parallel process"
  [(methodList size) < methodNumber] whileFalse:[
    m1:= (methodList at: methodNumber) asArray.
   (m1 at: 1) =3  ifTrue:[ aSemaphore wait ].   "-- (f) reaction for event"
          :
   (m1 at: 1) =6  ifTrue:[model recieveMessage: processID ].
                  methodNumber:= methodNumber + 1]]!
```

図 42: プロセス生成部の Smalltalk プログラムの例

4. ProcessBrowser(図 41 (d) 部)
   このクラスは，生成された ProcessObject の一覧や，ProcessObject 相互で行われる情報通信の状況，実行中の各 ProcessObject の状況の表示を行う．Application のサブクラスとして定義する．

6.3.3 プロセス実行管理機能の実現

6.2.2で定式化し，課題として列挙したプロセス間の交信とプロセスの生成や消去の管理メカニズムは，次の形で実現する．以下に課題とした項目ごとに，管理機能の実現の方法について述べる．また図 42に Smalltalk プログラムのプロセス生成に関する部分の一部を示し，プロセスの生成に関しては，この図の該当する部分を示しながら説明を行う．

プロセスの生成，識別子の一意性を確保した上でのデータベースへの登録 ProcessManager は，IdentityDictionary のサブクラスとして定義されている．このクラスは，一般的なプログラム言語の配列に相当し，同じ種類のクラスからインスタンス化されたインスタンスを複数内部に持つことができる (図 42の (c) 部)．このクラスは，Dictionary 内部に持つ複数のクラスインスタンスに対して同じメソッドを繰り返し適用したり，それらのインスタンスを取り出す，また消去させるなどの要素操作機能を持っている．ProcessManager は，この IdentityDictionary ク

ラスのサブクラスとして定義することで自身の配列に格納した要素の操作メソッドを継承し，ProcessObject の制御に関するメソッドを新たに記述することで定義される．

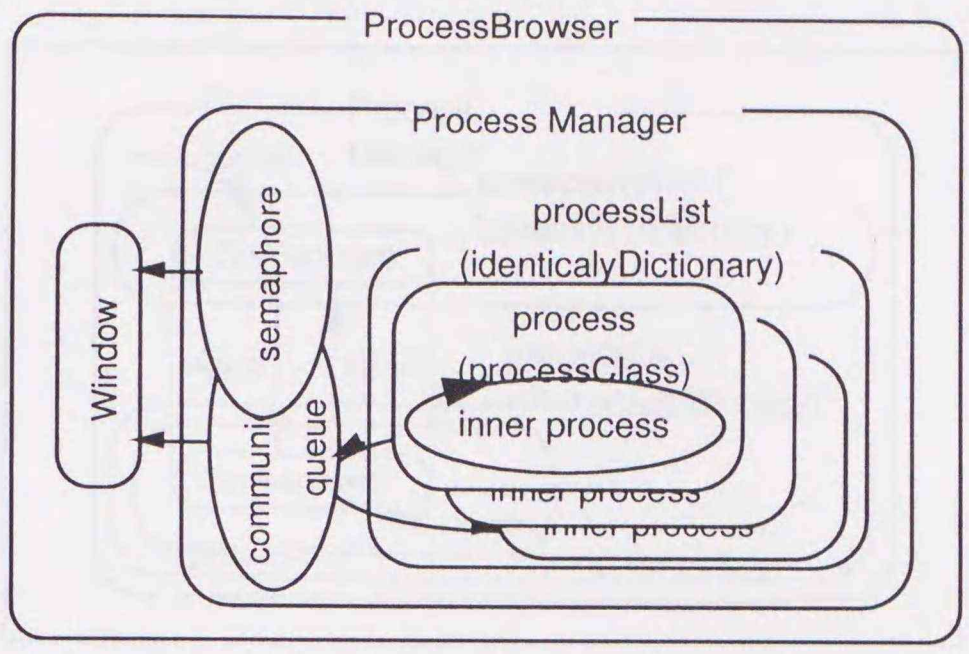


図 43: プロセスの生成

プロセスの平行動作の実現 ProcessObject はインスタンス化された後 (図 42の (d) 部)，引き数として与えられたアルゴリズムを子プロセスとして実行する fork メソッドを起動する (図 42の (e) 部)．インスタンス化された各 ProcessObject は，ProcessManager から見ると独立した普通のインスタンスとして扱うことができるが，その内部では予め定義されたプロセスが並行して動作している形になる．このアルゴリズムの部分に各プロセスでのイベントに対応する反応を定義する (図 42の (f) 部)．

他のプロセスからの要求に対応するプロセスの消滅，データベースからの削除 活動している複数のプロセスは，ProcessManager の内部変数の ProcessObject のインスタンスの内部で活動している．従ってインスタンスが消滅すれば，その内部のプロセスも消滅する．他のプロセス又は入出力対処から当該プロセスの停止を受け取った場合は，当該プロセスに対して一度停止のメッセージを発信して，当該プロセスが終了イベントに対する内部動作を終了させた時点で，再びメッセージを ProcessManager に送り返して，当該プロセス中の内部プロセスは活動を中止する．内部プロセスが停止しても ProcessObject のインスタンスは存在し続けるのでその消去は ProcessManager が行わなくてはならない．

並行活動しているプロセスと ProcessManager の通信の実現 ProcessManager とプ ProcessObject 内部のプロセスは，それぞれ並行して独立に動作しているので，プロセス同士で入出力を行う際には同期を取る必要がある．このために各 ProcessObject は，自身の入力待ちの状態に割り込みをかけるセマフォを ProcessManager と共有して，相互に入出力の割り込みが行えるようになっている (図 42の (c) 部)．各 ProcessObject と ProcessManager は，通常は入力待ち状態で待機している．この ProcessObject に対する情報入力は，本来送るべきメッセージ，引数，及び入力対象を ProcessManager に対して渡し，入力要求を出すことで行う．ProcessManager

は値引き渡し用のバッファにメッセージと引数とを格納して，対象となる ProcessObject のセマフォに対して割り込みを行う．割り込みを受けた ProcessObject は，メッセージと引数とを受け取り，あらかじめ定義された対応をとる．

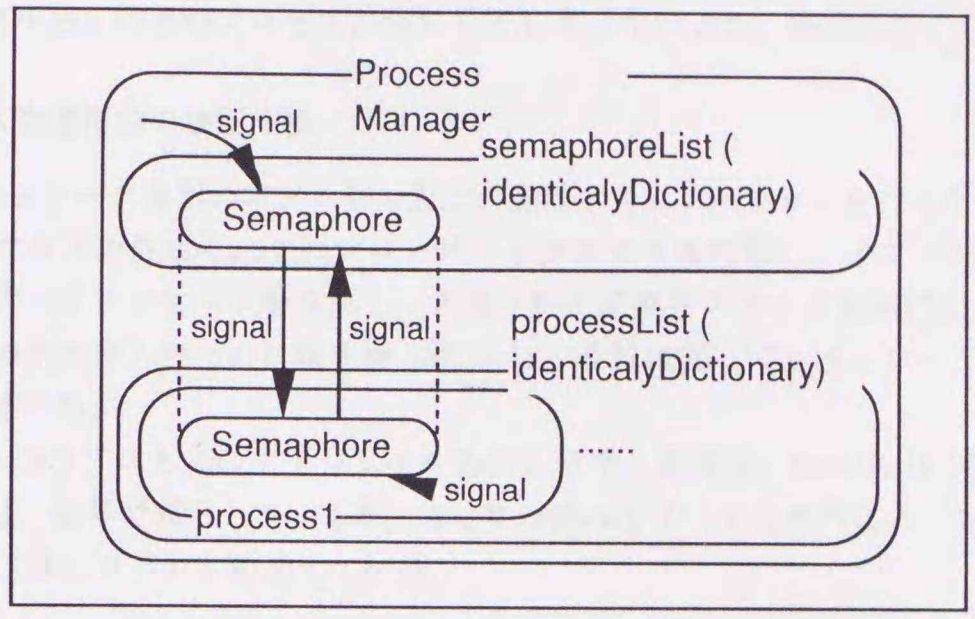


図 44: プロセスと ProcessManager の通信メカニズム

複数のプロセス集合を指定された際の同じメッセージの発信 複数の ProcessObject の集合を指定された場合，ProcessManager は引数として与えられた ProcessObject の集合の要素である一つ一つの ProcessObject に対して，単一の ProcessObject に対してのメッセージの発信を行うメソッドを自身に対してその個数の回数だけ呼び出すことでメッセージを発信する．

プロセスのタイプを指定された際の同じメッセージの発信 メッセージを発信する ProcessObject の集合ではなくて，タイプを指定された場合は，ProcessManager は初めに指定された自身が管理している identicalyDictionary の中から指定されたタイプの ProcessObject を抽出して，メッセージを発信する相手の集合を新たに作り出す．その後の操作は，前項のプロセス集合に対するメッセージの発信と同じメソッドによって行われる．

他のプロセスからの要求によるプロセスの活動状況の確認 ProcessObject の内部動作と ProcessObject のインスタンスとしての存在は独立したものであるので，ProcessObject 内部で活動する並行プロセスが停止してしまってもインスタンスは存在し続ける．EPDL でのプロセスは，他のプロセスをそのインスタンスへのポインタとしてしか指示できないので，入出力を行う対象の ProcessObject の内部で即に内部プロセスが停止してしまっていることが考えられる．しかし，ある ProcessObject が他のプロセスに対してメッセージを送ったとしても，相手の ProcessObject の内部状態によっては返事のメッセージが帰ってこないことが考えられる．従って，ある ProcessObject から見た場合に，単にメッセージを送るだけでは他の ProcessObject の内部で実際に活動しているかどうかを判断することができない．

このような状況に対処するため，設計生産プロセス記述言語では，プロセスの活動状況を判断する関数を設けている．この仕様を実現するために ProcessManager は ProcessObject の活動状況を他の ProcessObject の動作状況をアナウンスする機能を持っている．各 ProcessObject

は自身の内部プロセスが活動中であるかどうかを示す論理型の内部変数を持っており，内部プロセスが停止してしまった後は，このフラグ変数を false にする．セマフォを使ったメッセージの送信では，相手の ProcessObject の動向を把握することができないので，ProcessManager は通常のメソッドの送信で，相手 ProcessObject の内部状態変数の値を見ることによって，その ProcessObject 内部のプロセスが停止してしまっているか，停止していないかを判断する．

6.3.4 プロセス管理機能の検証実験

開発したシミュレータを用いてプロセス同士の通信シミュレーションを行った．実行した状況は，初めに一つのプロセスを生成し，そのプロセスが子プロセスを起動し，子プロセスが生成されて初期化されるのを待ってメッセージを交換し，入力された結果をブラウザ画面に出力するものである．各オブジェクトの生成や入出力の状況を図 45に示し，番号に従ってシミュレータ内の動作と入出力の状況に関して述べる．

この際の，２つのプロセスのスケジュールを次に示す．実際は，Smalltalk プログラムのソース中で各イベントは，番号で表現しているが，ここでは読みやすくするためにイベント名に置き換えている．また，--以降にコメントを補っている．

```
Process1 のスケジュール
((StartChildProcess 2)    -- プロセス２の起動
 (PushMessageFor 2 ' Hellow ' ' Process1 ' ' Process2 ')
                          -- プロセス２へのメッセージのスタックプッシュ
 (SendSignalFor 2)        -- プロセス２へのメッセージの発信のリクエスト
 (TimeDelayFor 1)         -- １秒の時間待ち
 (aSemaphoreWait)         -- セマフォからの入力待ち
 (RecieveMessageFor 1))   -- 自身へのメッセージのスタックからの受け取り

 Process2 のスケジュール
((aSemaphoreWait)         -- セマフォからの入力待ち
 (RecieveMessageFor 2)    -- 自身へのメッセージのスタックからの受け取り
 (PushMessageFor 1 ' Thank You ' ' Process2 ' ' Process1 ')
                          -- プロセス１へのメッセージのプッシュ
 (SendSignalFor 1))       -- プロセス１へのメッセージ発信リクエスト
```

上記のスケジュールに従ったプロセスマネージャと，各プロセスの動作は次の形で実行される．その状況を図 45に示す．

1. ProcessBrowser のインスタンス化 (図 45の (1))
   初めに ProcessBrowser がインスタンス化されて，内部変数である各ウィンドウの初期化を行う．

2. ProcessManager のインスタンス化 (図 45の (2))
   ProcessBrowser の内部変数として ProcessManager が初期化され，ブラウザから全体の処理を引き継ぐ．

3. ProcessScheduler の読み込み (図 45の (3))
   ProcessManager は，ProcessScheduler をインスタンス化し，初めに起動すべき ProcessObject

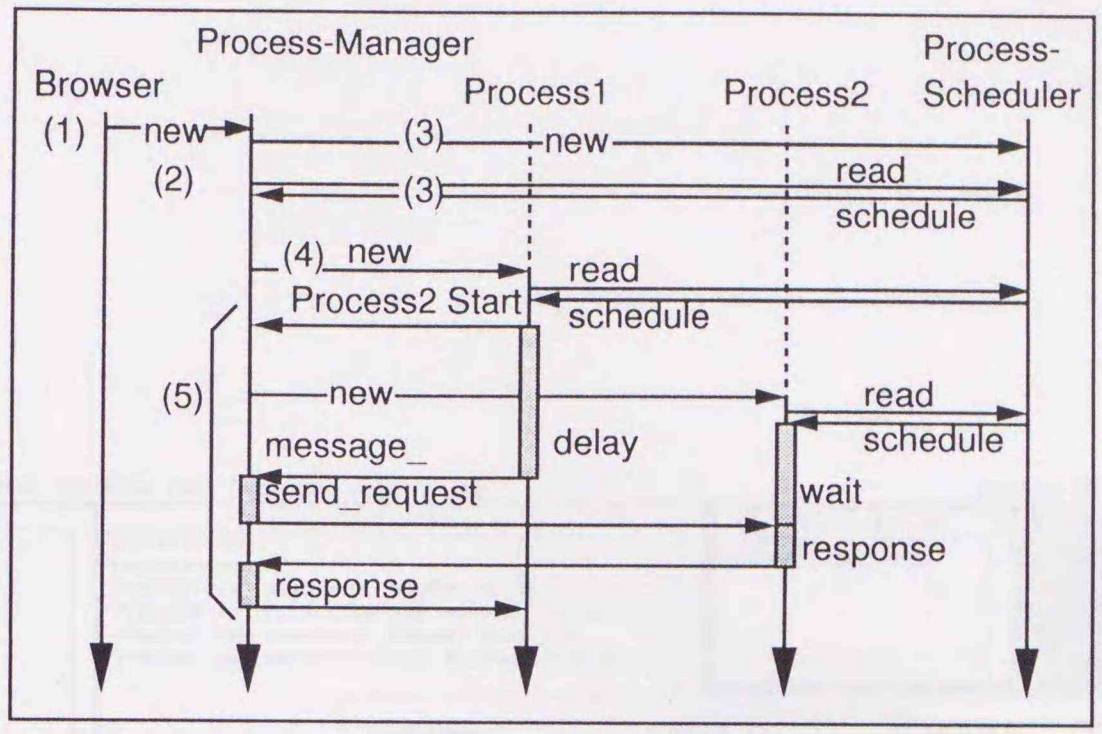


図 45: シミュレーションシステムの実行状況

の情報を受ける.

4. プロセスの生成 (図 45の (4))
   ProcessManager は，受け取ったプロセス情報に従って，初めに起動するべきプロセス Process1 を生成して自身の順序集合に格納する.

5. プロセス間通信の状況 (図 45の (5))
   起動された Process1 は，子プロセス Process2 を起動して，指定された時間動作を停止する. Process2 は，起動されるとイベント入力を受け付けるために，セマフォに対する割り込みの待ち状態に入る. Process1 は，待ち時間が終わった後で，メッセージを添えて ProcessManager にメッセージの発信を依頼する. ProcessManager は引数を入出力バッファに待避させて，待ち状態の Process2 のセマフォに割り込みを行う. 待ち状態のプロセスは，セマフォへの割り込みによって活動を再開して，入力されたメッセージを受け取り，逆の順序で Process1 にメッセージを返信する.

画面上に示された入出力の状況の例を図 46に示す. 図 46の上側のウィンドウは，システム内部で生じたイベントのリスト，下側のウィンドウはシステム内部で稼働中のプロセスのリストを示している.

上記の実験により，開発したシミュレーションシステムでの次の機能の実装が検証できた.

- 複数プロセスの発生と消滅の管理
- プロセス間の通信の管理と記録

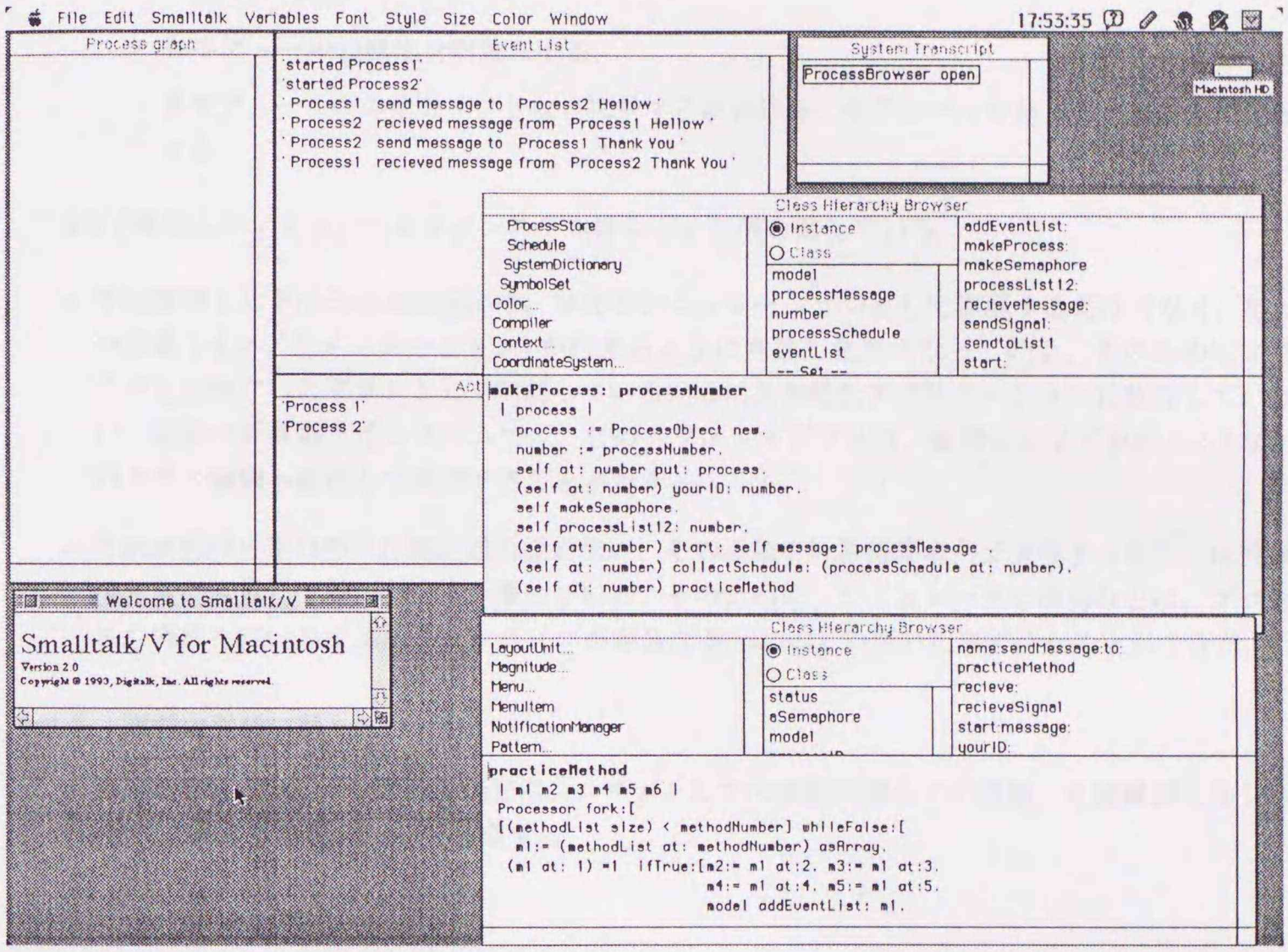


図 46: シミュレーションシステムの画面出力例

6.4 特徴及び他のモデルとの比較

以下では，本論文における計算機支援システムの特徴，他の研究との比較，第5章で提案したモデルの実装を考えた場合の具体的な実装などに関して考察する．

6.4.1 本論文におけるモデル構造の特徴

6.2で提案した計算機支援システムの一般系では，以下の特徴を備えている．

- システムの構成をモデュールに分けて提案している．このことによって次の特徴を得ることができる．
  - 各モデュールの機能を明確にする．
  - 各モデュールでの他のアプリなどに対する依存性を，モデュールを差し替えることで評価する．

6.3で提案したシミュレーションシステムは以下の特徴を備えている．

- 今回開発したProcessManagerは，単なるシミュレーションとして実現するだけでなく，種々の実装(インプリメンテーション)が行えるように考慮した形になっている．そのためにブラウザとは独立したクラスとして定義しているために具体的なアプリケーションに依存していない．実際の計算機支援システムでは，このクラスライブラリは，必要に応じて他のC++などのクラス階層へ変換して実装することができる．
- 活動表現は専用の形式仕様記述言語を用い，その活動を計算機環境内で実現する場合には計算機言語に変換して用いる方式を取っている．そのために，シミュレータの開発などは，プロセスの操作を行い易くまたプロトタイプが容易なSmalltalkを用いた形で行うことができた．

6.4.2 実際の実装に関しての考察

前章5で提案したモデルの計算機支援システムとしての実装に関しての課題，今回提案したシステムの拡張における課題について考察する．

今後のシステムの拡張

本論文で開発したProcessManagerの今後の拡張は，次の項目が考えられる．

1. ネットワークを用いた環境の中で，複数のProcessManagerを動作させる．
2. プロセススケジュールを読み込んで複数のプロセスの協調作業を実現する．
3. 実際にアプリケーションプログラムや設計活動者の情報の交換の制御を行う．
4. 設計生産における各種設備の入出力モデル化を行い，実際の情報入出力の制御を行う．

大きくは，ネットワーク環境での実装に関する課題，及び具体的な入出力対象を扱う場合の課題と考えることができる．以下では，これらの項目の中で1の複数ProcessManagerの動作，及び実装を考慮した具体的な入出力対象の実現に関して本論文での対応について考察する．

単一プロセスで動作する ProcessManager

6.3でのシステムの構成は，唯一つの ProcessManager が実装される形で提案された．しかしこの形式でシステムをネットワークを介した複数の計算機上に拡張した場合には，ProcessManager が動いている計算機へのネットワーク上のトラフィックが集中すること，及び ProcessManager 自体への負荷が集中する問題が起こる可能性がある．

この問題は，本論文で提案した基本モデルのプロセス代数に基づいて形式化を 3.2.2で行った際に，次の項目を満たす必要が生じたためであった．

- 特定のプロセスに対してあるいは入出力対象に対してメッセージを発信した場合に．起動されるプロセスの識別子の一意性を保つ．
- 複数のプロセスの集合が指定された際に，必要なプロセスのみに対してメッセージを発信する．
- 特定の種類のプロセスが指定された場合に，そのタイプの全てのプロセスに対して必要なメッセージを発信する．
- 特定のプロセスから，あるプロセスの消去を要求された際に，必要なプロセスのみを消去し，以降ではプロセスの識別子の一意性の保持を続けなくてはならない．
- 特定のプロセスが指定された場合に，そのプロセスが活動中であるかどうかを特定して要求を出したプロセス又は入出力対象に対して，返答を行う．

基本的には，上で列挙したプロセスの一意性に関する課題を複数の ProcessManager の間で確保できれば，システム全体の中で複数の ProcessManager を実行させることが可能と考えられる．このためには，上記のプロセス操作や通信に関するイベントが発生するごとに ProcessManager 間でプロセスの一意性の維持や，メッセージの配布の負荷の分散を行うメカニズムを用意する必要がある．

この場合には，ProcessManager を複数化した形でのプロセス代数による定式化を行う必要があると考えられる．

具体的な入出力対象の実現

今回開発した ProcessManager の管理機能を拡張して，計算機支援システムの開発を行う際には，外部入出力対象として記述したものを具体的にどう実現するかが課題となる．ProcessObject のサブクラスとして，特定のプロセスをモデル化した場合には，プログラム変換によって ProcessObject のサブクラスとしてモデル化すれば，そのまま用いることができる．しかし，入出力対象の場合は，入出力対象の内部活動は本論文の基本モデルでは対象とされていないので実装する必要はないが，その実際の対象に対して情報の入力を行う必要がある．

この場合には，入力対象に応じて次の対応が考えられる．

- 予め入出力の仕様が決まっているアプリケーションプログラムの場合
  - その入出力仕様を直接設計生産プロセス記述言語で表現する形式で定義し，実装の際には，その入出力仕様を直接呼び出す形に変換する．
  - CIMOSA における写像問題と同じく，概念的な設計生産プロセス記述言語による入出力仕様の表現から実際に行われる入出力機能への写像規則を決めておき，その規則に従った変換を行う．

- 設計者などの本来の入出力仕様が決まっていない対象
  - 概念的に入出力の仕様を定義しておき，写像規則を後から定義する方式.
  - 予め入力仕様を定義しておき，その組み合わせとして定義する方式.

対象への入出力仕様が予め決まっている場合と決まってない場合の違いはあるが，実装に関しての解決法方は，次の２つに絞られることになる.

- 予めその対象が持っている入出力仕様に合わせて設計生産プロセス記述言語で記述表現を行い直接変換する.
- その対象が持っている入出力仕様と１対１に対応していない，概念的な記述表現を行い，実装時に対応関係を考慮してプログラミングを行う.

次にモデル化の対象とされているプロセスが相手の入出力対象からの入力を受け入れる場合が課題となる．この場合には，まず特定の入出力対象からのメッセージを受信可能なプロセスの定義を設計生産プロセス記述言語において行い，多重継承を行うことによってその対象からの入力メッセージを受信できるようにする必要がある.

従って，実装を主眼とした仕様の表現を行う場合には，

- 特定の対象の入出力仕様
- 特定の対象からの出力を受信できるプロセスのモデル化

の両方を行い，リソースとして組み合わせる形式でのモデル化が重要となる.

3.2.2で定式化したとおり，メッセージの記述はその対象又はプロセスに対するイベントをモデル化したものであるので，設計生産プロセス記述言語によってモデル化を行う際に，対象がどのような性質のものであるか，どのような入力を受け付けることが可能であるかを熟慮する必要があると考えられる.

6.4.3 他の研究との比較

本来は 6.1.1 で調査した各研究ごとに 6.1.3に示した課題を満たしているかについての比較を行う必要があるが，各課題がそれ程詳細ではなく開発目標に近いことと，6.1.1 で上げた各研究が別個の目的に対するものであるので，本章では6.1.3の項目に捕われずで研究テーマごとに比較を行う.

CIMOSA [Esp93] [Ver94] [Ver93] による実装は，製造システムのオープン化を主目的にしているが，実際には CIMOSA が提案したアーキテクチャに従った専用工場システムによる実現に固執している．このため CIMOSA では，形式表現された概念的な工場のプロセスモデルから具体的な CIMOSA の工場仕様への写像が実装上の問題となっていた.

本論文では，生産システムの仕様定義と実装のような目的に絞ってはいないが，形式仕様記述言語で記述したモデルと具体的な実装との関係においては同じ問題が考えられた．この問題点について，本論文では対象への入出力仕様が確定しているかどうかで次の２つの解決法方を提案した.

- 予めその対象が持っている入出力仕様に合わせて設計生産プロセス記述言語で記述表現を行い，直接変換する.
- その対象が持っている入出力仕様と１対１に対応していない，概念的な記述表現を行い，実装時に対応関係を考慮してプログラミングを行う.

現実的には，実装を考慮したモデル表現を行う必要があることが判明した．

STEP に基づいた CAD システムのデータ交換 [ISO95b] [木村 95] [小島 93] [菊地 92] は，本来は製品モデル構造のみを行うための EXPRESS を，データ変換アルゴリズムの記述にまで拡張して用いていた．このため，これらの研究では EXPRESS による記述に対して，

- 計算機システム上での EXPRESS 表現されたデータ構造とアルゴリズムの実現
- そのデータ構造に対する別の CAD システムファイルからの入出力
- 計算機システム上でのデータに対する変換アルゴリズムの起動の指定

が課題となっていた．

これに対して本論文では，

- 計算機システム上での形式表現された入出力対象とプロセス，又特定のデータ構造に対するアプリケーションシステムファイルからの入出力は，それぞれクラス定義を行い通信や子プロセスの起動のメソッドを設け，そのサブクラスとしてユーザが定義したプロセスや入出力対象を定義する．
- 計算機システム上でのプロセスの起動に対しては，スケジュールデータに従って ProcessManager が子プロセスを起動する形で規定する．

などの方針で対応した．

LOTOS を生産活動表現に用いた報告 [BB93] では，

- シンタクスチェッカ，シミュレータ，テスター，ベリファイア，コンパイラなどの開発への方向付けを行う．

が課題となっていた．本論文では，上記のシステムの全てを実装することは不可能であったが，最も複雑と考えられる並行プロセスシミュレータが実現できたので他のシンタクスチェック関係での対応も可能であると考えられる．

6.5 結 論

本章での報告事項を次にまとめる．

- 計算機支援システムに必要な機能を次の２項目に沿って分類した．
  1. 記述表現の妥当性に対する判定結果や，記述内容の別形態への変換結果などの出力形式．
  2. 表現されたプロセスを実際にシステムの内部で実行させる際の実行形式．
- 設計生産活動の形式表現を扱う計算機支援システムの機能に関して調査を行い，CIMOSA のような専用システムでの実装は，形式記述表現から予め用意されているサービス関数への翻訳 (写像) をどのように規定するかが課題となることが指摘された．
- 本論文における課題として次の項目が上げられた．
  - 単なるシミュレーションとして実現するだけでなく，種々の実装 (インプリメンテーション) が行えるように考慮して，具体的なアプリケーションに依存しないクラスライブラリなどの形に必要に応じて変換して実装できる方式で開発を行う．

- そこで，活動表現は専用の形式仕様記述言語を用い，その活動を計算機環境内で実現する場合には計算機言語に変換して用いる方式が望ましい．

- 次の項目で計算機支援システムの提案を行った．
  - 課題となった機能を実現する計算機支援システムの基本的な構造と各部の機能を提案した．
  - 形式仕様記述言語による情報交換の状況を実現するためにシステム内部で実行されるプロセスの動作管理機能をプロセス代数に基づいて定式化した．

- 次の項目で具体的な計算機支援システムの開発に必要なプロセス管理機能の実現可能性の検証に関する報告を行った．
  - プロセスの動作管理機能を実現するシミュレータシステムの基本的なクラス階層構造を提案した．
  - Smalltalk/V を用いてプロセス実行管理機能の具体的な開発を行った．
  - プロセス間の通信の実験を行い，システム内部で同時並行プロセスが稼動し，プロセス管理機能を実装可能であり，本論文で提案した基本モデルと形式仕様記述言語に基づいた設計生産活動の計算機支援システムを実現可能であることを確認した．

- 次の項目で特徴，他の研究との比較，実際的な計算機支援システム開発についての考察を行った．
  - 本論文における計算機支援システム一般系では，システムの構成をモデュールに分けて提案している．このことによって 1) 各モデュールの機能を明確にすることができる．2) 各モデュールでの他のアプリなどに対する依存性を，モデュールを差し替えることができる．などの特徴があることを示した．
  - 今後のシステムの拡張と実際的な実装に関して検討し，1) 複数の ProcessManager を動作させる際の問題．2) 実際にアプリケーションプログラムや設計活動者に対する情報入出力の制御を行う．の２つの課題について更に考察を行った．これらの考察から，具体的な対象の表現と実装に関してはモデル化を行う時点からその対象が本来入出力できるメッセージ (イベント) を十分に把握してそのメッセージの直接のモデル化を行うか，仕様の記述と実装時の写像関係を予め規定しておく必要があることが判明した．
  - 他の研究との比較を行い，各課題に対しての本論文の対応における相違点を明示した．
    * 対象への入出力仕様が確定しているかどうかで次の２つの解決法方を提案した．1) 予めその対象が持っている入出力仕様に合わせて設計生産プロセス記述言語で記述表現を行い，直接変換する．2) その対象が持っている入出力仕様と１対１に対応していない，概念的な記述表現を行い，実装時に対応関係を考慮してプログラミングを行う．現実的には，モデル化の際に実装を考慮したモデル表現を行う必要があることが判明した．
    * 計算機システム上での形式表現された入出力対象とプロセス，又特定のデータ構造に対するアプリケーションシステムファイルからの入出力は，それぞれクラス定義を行い通信や子プロセスの起動のメソッドを設け，そのサブクラスとしてユーザが定義したプロセスや入出力対象を定義する．

* 計算機システム上でのプロセスの起動に対しては，スケジュールデータに従ってProcessManagerが子プロセスを起動する形で規定する．
* シンタクスチェッカ，シミュレータ，テスター，ベリファイア，コンパイラなどの開発への方向付けは，本論文では，上記のシステムの全てを実装することは不可能であったが，最も複雑と考えられる並行プロセスシミュレータが実現できたので他のシンタクスチェック関係での対応も可能であると考えられる．

7 結論

本論文は，設計生産の活動に関する形式的な表現を扱う計算機支援システムのための活動モデルと，その表現方法，計算機支援システムの開発方法論を提案することによって，設計生産活動の計算機による管理と活動に関する情報の規格化を可能とさせることを目的とする．本論文では，このような目的に対して，設計生産の活動と活動に関与する対象を表現するための基本モデルの確立，その基本モデルに基づく形式仕様記述言語 (設計生産プロセス記述言語) の開発，言語仕様に基づいた並行活動モデル (設計生産活動統合モデル) の提案，計算機システムの開発などに関する報告を行い，以下に述べる結論を得た．

第1章では，背景としてコンカレントエンジニアリングの概念に基づいた設計活動と生産活動の並行した進行と，世界的規模での設計生産分業体制への移行，設計生産の分野での CAx システム (Computer Aided x System) の発展によるデジタルデータによる設計生産活動への移行，などの環境変化について述べた．次にこの環境変化によって顕著となった問題点を明らかにした．

- 従来方式での管理業務は，地理的に分散されて行われる設計生産活動では困難になる．
- 印刷物としてしか明文化されていない管理業務の規則と設計生産活動の成果であるデジタル化されたデータとの不整合のために，活動全体の効率を上げることができない．
- PL 法対策などのための，製品を保証するための情報として，製品の開発過程や製造過程に関する記録を計算機可読な形で保存することができない．
- 類似した設計生産活動で過去の事例の記録を自動的に検索することが困難．

これらの問題点に対して既存の研究である PDM と SADT 手法に関する分析を行い，本研究の対応として

- 設計生産の活動に関する情報を蓄積／共有し，その内容に従う活動の質を保証するための基準を規定する．
- 設計生産の活動に関する情報を計算機内部で取り扱い，その意味的な妥当性を判定したり，直接計算機支援システムを実現するための方法論の確立．

からなる解決法を提案した．より具体的には，次の方法論を取ることとした．

- 設計生産の活動に関する情報を，人間と計算機とが取り扱える形で客観的に表現するための全体モデル設計生産コミュニケーションモデル (DMCM : Design and Manufacturing Communication Model) を確立する．即ち
  - 設計生産の活動を表現する形式仕様記述言語を用いて表現する基礎となるための基本モデルを確立する．
  - 形式仕様記述言語を開発する際の仕様を決める基準とするために，提案した基本モデルが動作する世界を完全に規定するために，提案した基本モデルをプロセス代数を用いて定式化する．
  - 設計と生産活動を表現することに特化した，形式仕様記述言語の開発を行う．

  - 大規模なモデルを構築する方法論を確立する.
  - 計算機による実装技術を確立する.

- 提案した表現技術に従って，設計部門から生産部門までの活動を表現する活動モデル 設計生産活動統合モデルを構築する.
- 提案した形式仕様記述言語を用いる計算機支援システムの一般的な仕様を提案し，その実現のために重要と考えられるプロセス管理機能の実装を試み，シミュレータとして実現する.
- 基本モデルと形式仕様記述言語の機能に関して，他の研究との比較を行う.

第 2章では，第 1章であげた目的に対する問題点を個別の項目に整理して分析を行った．その結果，本論文が解決すべき課題として次の問題点を得ることができた.

- 設計生産における活動に関与する，設計者や設計グループ，CAD/CAM システムやエンジニアリングデータベースなどのアプリケーションプログラム，複数の工作機械やロボットなどの製造設備，製造設備や貯蔵設備などで操作の対象となる製品，素材及び工具などの部品類，入出力されるデータ，データの構造に関するデータ (メタデータ) などを表現対象として取り扱う必要がある.
- 上で挙げた様々な対象の間で，同時に並行して進行する複数の活動を表現するための基本モデルを確立する必要がある.
- 設計生産に関係する表現対象と，その間で実行される活動を表現するための基本モデルを確立したのちに，そのモデルに基づく原子的な意味を規定し，それらを予め期待された構文に従って組み合わせることで表現の対象のモデルを規定する，形式仕様記述言語を確立する必要がある.
- 上で挙げた基本モデルと形式仕様記述言語を用いた上で，対象とされる各種の活動を表現するモデルを構築するための，基本的な構成要素を組み合わせて，複数の活動を関与させることができるモデル構造を確立する必要がある.
- 上で挙げた基本モデル，形式仕様記述言語，モデル構造に従った設計生産活動に関するモデルを計算機システム又は他の生産システムで用いるための方法論の確立に対する必要がある.

このように設計生産の活動に関して，単純に個別のモデルを提案することに注目せずに，初めにそのモデルを形式仕様記述言語を用いて表現する方法論を確立し，モデルの表現を行う階層的な方法を取ることによって，部分的に別の性質のモデルなどを組み合わせることが可能である.

第 3章では， 1.2で述べた設計生産の活動を表現する際の問題点，及び 2.1の設計生産活動の表現のための 基本モデルに関する問題点の分析に対して，それらの問題点を考慮したモデルの提案を行い，そのモデルを 設計生産コミュニケーションモデルにおける基本モデルと名付けた．第 3章の結果を次にまとめる.

設計生産活動を表現する既存の研究での概念に関する分析を行い，これまで用いられてきた方法論の分析を行った.

本論文で目的としている設計生産活動を表現するためには，特定のモデルに対する依存性が少ないこと，動的な活動を表現する必要があること，多様な対象を表現する必要があること，高レベルの

情報を交換する必要があること，プロセスの実行メカニズム考慮した設計にする必要があることなどが判明した．

設計生産活動の表現以外の分野での研究として，オブジェクト指向，プロセス代数及び通信プロトコル記述言語などの研究が行われており，メッセージを受信することでメソッドが起動されるメカニズムや，プロセスを「周囲のプロセスや周りの環境と通信を行う抽象的なエージェント」としてとらえる考え方，プロセス代数における情報の入出力の観察結果をプロセスとして扱う概念が本論文の目的に適していることを明らかにした．

本論文で設計生産に関する活動を表現するための基本モデルの提案を行った．これはオブジェクト指向的な考えに加えて，情報入出力の観測結果をプロセスとして扱うプロセス代数の考え方を元に，設計生産の活動において種々の対象の間で特定の活動に関する情報をやり取りする概念的な媒体として設計生産におけるプロセスを表現するものである．

設計生産におけるプロセスを，特定の活動に関する情報の入出力の過程として表現するために，そのような活動に関与する様々な対象を，情報入出力の対象として表現することを提案した．

本論文で提案したプロセスと入出力対象に関する基本モデルは，プロセス代数の CCS に基づいたものである．入出力における各プロセス動作を CCS に基づいて表現するために，活動中の全てのプロセスの識別子の集合を持ち，プロセスの生成や消滅を実際に行うプロセスマネージャの概念を導入した．プロセスマネージャの概念を用いることによって，プロセスから一つのプロセス，プロセスの集合，特定の種類のプロセスへの情報の入出力．プロセスの生成，終了，他のプロセスからの強制的なプロセスの停止，プロセスが活動しているかどうかの判定などの，プロセスの動作を表現する基本モデルを表現することができた．ここで表現されたプロセスマネージャの各機能は，本論文の基本モデルを実装する際の，重要な課題となる．

本論文で提案したプロセスと入出力対象に関する基本モデルは，オブジェクト指向的な考えでは，まず抽象的な入出力の対象のクラスを定義し，その下位型として本論文で提案したプロセスと入出力対象をモデル化する．実際の設計生産活動モデルで用いられる活動プロセスや，CAD/CAM システムや設計活動者などの具体的な入出力対象は，これらのプロセスと入出力対象との下位型としてモデル化される．

本論文における基本モデルの特徴は，複数のプロセスが並列して動作し，それぞれ独立して情報の入出力行うことによって並列活動を表現することができること，並行動作するプロセス同士の時間的相互関係は，特定のプロセスが次のプロセスを起動する際の時間的な前後関係として表現できること，ルーリベースシステム以外の，多様な考え方のモデルの表現が行えること，設計生産に関与する様々な対象に対する広い表現能力を備えること，この概念に合わないモデルもブラックボックスである入出力の対象として表現することで，複数の概念に基づいたモデルも組み合わせることができる，などがある．

3.1.2でまとめた課題に対して他のモデルとの比較を行い，並列プログラミング言語と同様に本論文で提案した基本モデルが有効であることを明らかにした．

本論文で提案した基本モデルでは，情報以外の物の移動を直接表現することが難しいこと，単純に並行活動するプロセスを多数記述するだけではモデルの複雑化が懸念されること，安定性の数学的な解析が難しいこと，ルールベースシステムの表現が難しいこと，などの限界が明らかになった．

基本モデルでは完全に表現できないため，形式仕様記述言語の課題として時間的な情報を取り扱う必要があること，動作している同種のプロセスの識別が一意にできる仕様が必要であること，入出力される情報が単なるシグナルレベルでなく，より高度な情報であることが必要である点が判明した．

第4章では，1.2で述べた設計生産の活動を表現の問題点について，また2.2において行った設計生産における活動を表現する基本モデルに基づく形式仕様記述言語の問題点に対して，それらの問題点を考慮して設計生産活動に関する形式仕様記述言語の提案を行った．その言語を設計生産プロセス記述言語 (Engineering Process Description Language : 以下EPDLと呼ぶ) と名付けた．第4章の結果を次にまとめる．

設計生産活動を形式言語を用いて表現する既存の研究に関する分析を行い，これまで用いられてきた言語仕様の分析を行い，本論文で目的としている設計生産活動を表現するためには，次の項目が必要であることが判明した．

- 言語要素を少なくして仕様の複雑化を避けること，
- 独立した動的なプロセスを表現し，相互の強調を表現すること．
- 入力ステートメントを用いないで，イベントループを言語仕様として表現すること．
- 入出力されるデータをユーザが規定できるようにすること．
- 生産過程での入出力に製品などの物流を表現すること．
- 予め定義されて変化しない通信構造を用いない．
- データとしての参照と情報入出力の構造が交錯してモデルが複雑化を避けること．

上記の要件に対して，設計生産に関する活動を表現するための設計生産プロセス記述言語の提案を行った．これは，3.2.1で提案した基本モデルに基づくもので，設計生産の活動における種々の対象の間で特定の活動に関する情報をやり取りする概念的な媒体としプロセスを表現する．またそのような活動に関与する様々な対象を，情報入出力の対象として表現するものである．

設計生産プロセス記述言語の特徴は，イベントドリブン形式を用いているので，ユーザがイベントループを記述する必要がないこと，同じ理由からプロセスの動作を，メッセージ毎の形で表現できること，これによってユーザは，入出力されるメッセージの表現に傾注できること，などがある．

次に 4.1.2でまとめた課題に対して他の言語との比較を行い，本論文で提案した基本モデルが有効であることを明らかにした．

また，本論文で提案した言語仕様では情報以外の物の移動を直接表現することが難しいため物流の表現に関しては，

- 移動する物体を情報として各対象の間を入出力させる．
- 移動物体も入出力対象として表現する．

の２つの方法で対応できることを示した．

さらに本論文での課題として，特定時間における通信構造の表現を行なう必要性が考察された．また発展的な研究として，パラメトリック形状モデラの研究に結び付けられることを述べた．

第5章では，1.2で指摘したモデルの構築方法の問題点，及び2.3において行った大規模なモデルの構築に関する問題点の分析などを考慮し，第3章及び第4章で提案した基本モデルと形式仕様記述言語に基づいた設計生産活動モデルの基本構造を提案し，その基本構造に基づいた設計生産活動モデルの構築を行った．第5章の結果を次に述べる．

本論文のモデリングの対象として考えられる設計グループ活動モデルや生産活動モデルに関する調査を行い，表現対象や表現される活動，モデルの構造に関して次の項目での問題点が明らかになった．

- 設計者，設計者のグループ，グループの管理者，データを入出力するCADシステム，データを貯蔵するデータベースなどを設計活動で関与する対象として考える必要がある．
- 設計者グループは一般的には製品の構造によって，機械系，電装系などに分けられ，その中では部品の階層構造に応じた階層構造を取っている．
- 設計者グループは，常に決まった組織で運営されている訳ではなく，その組織構造は仮想的なものであり，設計活動の進展に応じて組織構成が変更されると考えられる．一つのプロジェクトに複数の作業者が関係することは無論のこと，個人の設計者が複数の設計プロジェクトに参加することも考えられる．
- 生産活動に関与する対象として，ロボット及び工作機械などの生産設備，AGVやコンベアなどの輸送設備，実際に加工される被加工物や工具などを表現する必要がある．
- 部品など一般的な製品の入出力として，設計生産において加工される実際の製品，工作機械で用いられる工具，活動に費やされる人的な資源の表現，
- 多様な入出力対象表現として，EXPRESSによるデータモデルの記述やSTEP物理ファイルなども情報入出力の対象とすることによって，STEPにおける製品モデルとのインターフェースを持必要がある．
- この対象による活動として，通常の設計データの入力や修正の活動，データが完成した後での公表，承認作業が考えられる．チームリーダーは，活動，活動のゴール，デッドラインなどに関する連絡を可能にし，チームメンバーによる活動の進展に密着し続け，個別の決定の効果をチーム全体に対する公表を可能にする必要がある．
- 設計データの参照，設計データの修正変更，設計グループ構造などに関する種々の権利を設定する必要がある．
- 活動(各設計プロセス)は，階層化，構造化される必要がある．また相互の関係を表現する必要がある．
- 生産設備を用いる活動状況や搬送作業を表現する必要がある．
- 一定のモデル構造を確定しておき，その構造に従って要素モデルを組み合わせてモデルの構築を行う手法を用いる必要がある．
- 基本的な情報入出力の構造，対象，及び過程に関するモデルを予め用意しておき，その内容をオブジェクト指向と同じく継承して再定義することによって，個別の設計生産活動モデルを構築する方法が有効と考えられる．
- 常に決まった組織形態を維持するのではなく，設計活動の進展に応じて仮想的な組織が構成され，その中で各種の活動が行われる状況の表現．

上記の問題点に対して，本論文で設計生産に関する活動を表現するための設計生産活動統合モデルの提案を行った．これは上に挙げた諸問題を克服して，設計生産活動における種々の対象と仮想的な組織を表現し，その対象間で実行される活動を表現するために，基本的な入出力対象とプロセスのモデル化を行い，それらを継承して設計生産活動モデルの構築を行うものである．

設計生産活動統合モデルの特徴は，特定の活動において起こるイベントを一種類のプロセスとしてモデル化し，スケジュールなどのデータに応じて個々の活動に相当するプロセスを起動して情報の入出力を制御する．階層的な構造を作る入出力対象のモデルを用いることで，仮想的な活動グループの構造を任意に作り出すことができる，などがある．

5.1.1及び5.1.3 でまとめた課題に対して他のモデル化方法論との比較を行い，本論文で提案したモデル構造が有効であることを明らかにした．

全く異なった目的の活動が相互に関与する状況 (例えば搬送と加工) を表現する方法に関して考察を行った．一度，別々にモデル化を行い，組み合わせた活動として表現する場合には，多重継承を行って複数の活動に関するイベントを受け付けることを可能にし，スケジュールからは一種類のプロセスとして扱えるようにすることによるモデル化の方法を示した．

第6章では，形式記述表現された設計生産活動を用いる計算機支援システムについて述べる．この分野では具体的な方法論が確立されていないので，計算機応用システムで必要とされる機能の定式化に重点を置いて報告を行った．第6章での結論を次にまとめる．

計算機支援システムに必要な機能を，1) 記述表現の妥当性に対する判定結果や，記述内容の別形態への変換結果などの出力形式．2) 表現されたプロセスを実際にシステムの内部で実行させる際の実行形式．の２項目に沿って分類した．

設計生産活動の形式表現を扱う計算機支援システムの機能に関して調査を行い，CIMOSA のような専用システムでの実装は，形式記述表現から予め用意されているサービス関数への翻訳 (写像) をどのように規定するかが課題となることが指摘された．

本論文における課題として，1) 単なるシミュレーションとして実現するだけでなく，種々の実装が行えるように考慮して，具体的なアプリケーションに依存しない，クラスライブラリなどの形に，必要に応じて変換して用いる方式で開発を行う．2) そこで，活動表現は専用の形式仕様記述言語を用い，その活動を計算機環境内で実現する場合には計算機言語に変換して用いる方式が望ましい．２点が上げられた．

次の項目で計算機支援システムの提案を 1) 課題となった機能を実現する計算機支援システムの，基本的な構造と各部の機能を提案した．2) 形式仕様記述言語による情報交換を計算機システム上で実現するために，システム内部で実行されるプロセス管理機能を，プロセス代数に基づいて定式化した．の２点に関して行った．

具体的な計算機支援システムの開発に関する報告を次の３点に関して行った．1) プロセスの動作管理機能を実現するシミュレータシステムの基本的なクラス階層構造を提案した．2) Smalltalk/V を用いて，プロセス実行管理機能の具体的な開発を行った．3) プロセス間の通信の実験を行い，システム内部で同時並行プロセスが稼動し，プロセス管理機能を実装可能であり，本論文で提案した基本モデルと形式仕様記述言語に基づいた設計生産活動の計算機支援システムを実現可能であることを確認した．

特徴，他の研究との比較，実際的な計算機支援システム開発についての考察を行い次の結論を得た．本論文における設計生産活動モデルの一般系では，システムの構成をモデュールに分けて提案している．このことによって 1) 各モデュールの機能を明確にすることができる．2) 各モデュールでの

他のアプリなどに対する依存性を，モデュールを差し替えることができる．などの特徴があることを示した．

今後のシステムの拡張と実際的な活用に関して検討し，1) 複数のプロセスマネージャを動作させる際の問題．2) 実際にアプリケーションプログラムや設計活動者に対する情報入出力の制御を行う．の２つの課題について更に考察を行った．これらの考察から，具体的な対象の表現と実装に関してはモデル化を行う時点からその対象が本来入出力できるメッセージ (イベント) を十分に把握してそのメッセージの直接のモデル化を行うか，仕様の記述と実装時の写像関係を予め規定しておく必要があることが判明した．

他の研究との比較を行い，対象への入出力仕様が確定しているかどうかで次の２つの解決法方を提案した．1) 予めその対象が持っている入出力仕様に合わせて設計生産プロセス記述言語で記述表現を行い，直接変換する．2) その対象が持っている入出力仕様と１対１に対応していない概念的な記述表現を行い，実装時に対応関係を考慮してプログラミングを行う．

現実的には，モデル化の際に実装を考慮したモデル表現を行う必要があることが判明した．

以上本論文を総括すると，設計生産活動の計算機支援システムを構築するために，設計生産に関係する種々の表現対象を情報を入出力する対象として考え，この対象間での情報の入出力現象を設計生産の活動として見る基本モデルを提案し，この基本モデルに基づいた形式仕様記述言語を用いた活動表現による活動表現技法とを提案し，具体的な活動モデルと計算機支援システムの提案を行った．各章で，他の研究と本論文で提案した方法との比較を行い設計生産活動の表現とその活用に関する方法論の確立を行うことができた．

A　付 録：EPDL 言語仕様

範 囲

本文書は，EPDL の言語仕様と，その機能に関して述べる．

構文規則

構文定義は，以下の WSN 記法 (Worth Syntax Nortation) に従って記述する．

- 構文定義 (syntax) は生成 (production) の一ないしそれ以上のくり返しから構成される．
- 生成は，識別子 identifier '=' expression で構成される．即ち式の左辺の識別子の名前を持つ構文は，右辺の式で表現される．
- 式 (expression) は，項 (term) と or 記号"|"で構成される．即ち式は，term または term — expression で再帰的に表現される．
- 項 (term) は，要素 (factor) の一ないしそれ以上のくり返しから構成される．
- 要素 (factor) には，identifier , literal , group , option , repetition がある．
- 書き文字 (literal) は，そのままの状態で書きあらわされる文字を表現する．即ち'=' と記述があれば等号記号が示されることになる．
- グループ (group) は，式を一つのまとまりとして扱うために用いる．
- オプションは，かぎ括弧の中の式が 0 ないし 1 回出現することを意味する．
- くり返し (repetition) は，括弧の中の式が 1 ないしそれ以上出現することを意味する．

以下に，WSN 記法に基づいた WSN 記法を示す．

```
syntax     = { production }.
production = identifier '=' expression '.' .
expression = term { '|' expression } .
term       = factor { factor } .
factor     = identifier | literal |
             group | option | repetition .
literal    = '''' character { character } '''' .
group      = '(' expression ')' .
option     = '[' expression ']' .
repetition = '{' expression '}' .
```

特殊記号規則

下記の文字は，印字不可能な制御文字を表現する．

\a	任意の印字可能な文字
\n	改行文字
\q	quote 文字 (')
\s	空白
\o	任意の文字
\xx	16 進であらわされる数　xxh
\\	\ 自身を表現

基本言語要素

ここでは，EPDL に出現する言語要素を定義する．

文字セット

初めに基本的な文字集合を定義する．

1. 数字

```
digit          = '0' | '1' | '2' | '3' | '4' | '5' |
                 '6' | '7' | '8' | '9' .
```

2. 文字

```
lower_letter  = 'a' | 'b' | 'c' | 'd' | 'f' | 'g' |
                'h' | 'i' | 'j' | 'k' | 'l' | 'm' |
                'n' | 'o' | 'p' | 'q' | 'r' | 's' |
                't' | 'u' | 'v' | 'w' | 'x' | 'y' |
                'z' .

upper_letter  = 'A' | 'B' | 'C' | 'D' | 'F' | 'G' |
                'H' | 'I' | 'J' | 'K' | 'L' | 'M' |
                'N' | 'O' | 'P' | 'Q' | 'R' | 'S' |
                'T' | 'U' | 'V' | 'W' | 'X' | 'Y' |
                'Z' .

under_score   = '_' .

special       = not_paren_star_special | '*' | '(' | ')' .

not_paren_star_specal
              =  '!' | '@' | '#' | '$' | '%' | '^' |
                 '&' | '_' | '-' | '+' | '=' | '{' |
                 '}' | '[' | ']' | '~' | ':' | ';' |
                 '"' | ''' | '<' | '>' | ',' | '.' |
                 '?' | '/' | '|' | '\' .
```

3. 注釈

注釈は，'(*' と '*)' に挟まれた埋め込み注釈と，'--' で始まって改行コードで終わる行端注釈を用意する．

```
embeded_remark  = '(*' { not_lparen_star | lparen_not_star |
                  star_not_rparen | embeded_remark } '*)' .
not_lparen_star = not_paren_star | ')' .
not_paren_star  = upper_letter | lower_letter | digit |
                  not_paren_star_special .
lparen_not_star = '(' not_star .
not_star        = not_paren_star | '(' | ')' .
star_not_paren  = '*' not_paren .
not_rparen      = not_paren_star | '*' '(' .

tail_remark     = '--' { \a | \s } \n .
```

例 1.　正しい例

```
INTERFACE p1;
(* これは埋め込み注釈である *)
BEGIN

END_INTERFACE ; -- これは終端注釈である.
```

例 2.　正しくない例

```
INTERFACE p1;
(* これは埋め *) 込み注釈である *)

END_INTERFACE ; -- これは
終端注釈である.
```

4. シンボル

数式，演算子等で用いられるシンボルには以下の種類がある.

```
. , ; : * + - = \ / < > [ ] { } | ( ) **
<= >= <* := || % ' (* *) -- e   &
```

予約語

言語の中で特殊な意味を持つキーワードには，以下のものがある．それぞれがどのような構文で用いられるかは，以下の構文定義を参照すること．またキーワードと同じ識別子は認められない.

1. キーワード

```
ARRAY
BEGIN
BROAD_CAST
BY
CHAR
CLASS
DELAY
ELSE
END_CLASS
END_FUNCTION
END_INTERFACE
END_LOCAL
END_PROCESS
EPDL
ESCAPE
FUNCTION
IF
INHERIT_FROM
INTEGER
INTERFACE
LIST
LOCAL
MOD
NOT
OF
PROCESS
```

```
REAL
REPEAT
RETURN
SKIP
SUBCLASS
THEN
TO
```

2. 演算子としての予約語

数式中で演算子として用いる予約語には以下のものがある.

```
AND  ANDOR DIV  IN  LIKE MOD  NOT  OR  XOR
```

3. 標準定数としての予約語

定数として用いる予約語には以下のものがある.

```
TRUE       FALSE
```

4. 標準関数

標準関数として用いられる予約語には以下のものがある.

```
ABS  ACOS ASIN ATAN   BLENGTH  COS  EXP  LOG  LOG10
LOG2 ODD  SIN  SIZEOF SQRT     TAN
```

5. 型操作関数

EPDL は，型操作を行う第二階の操作能力を持たないが，受け取った変数の型を文字列として返す関数のみを用意する.

```
TYPEOF
```

6. プロセス操作関係の特殊組み込み関数

プロセス操作を行うための組み込み関数には，以下のものがある．各関数の機能と代えす値に関しては，以下の記述を参考にすること.

```
SELF_PROCESS
ABORT
IS_COMPLETED
IS_ACTIVE
IS_VALID
```

識別子

変数名，ブロック名，などに用いられるものが識別子である．識別子の構文は，頭がupper_later またはlower_letter で，アンダースコアまたは数字またはupper_later またはlower_letter が任意の個数続く.

```
simple_id     = ( upper_letter | lower_letter )
                { upper_lower_under | digit } .

qualified_id  = { class_id '.' } simple_id .

upper_lower   = ( upper_letter | lower_letter ).
upper_loser_under
              = ( upper_letter | lower_letter | under_score).
```

例 3.

正しい例
abcde
Abcde
Abcdef456

cylinder1.stroke

正しくない例
9avd
abf%efg

リテラル

リテラルは，数式中である値を表現するために用いられる．内容は整数リテラル，実数リテラル，文字列リテラルがある．

```
literal   = integer_literal | real_literal | string_literal
            logical_literal | complexed_literal .
```

1. Integer Literal

整数値を表現する整数リテラルは，10 進数の数字の羅列によって表現される．

```
integer_literal = digits .
digits          = digit { digit } .
```

例 4.

正しい例
1
0123

正しくない例
.12

2. Real Literal

実数値を表現する実数リテラルは，10 進数の実数部と，オプションとしての小数部および架数部から構成される．架数部はオプションとしてさらにサインを持つ．

```
real_literal  = digits '.' [ digits ]
                [ 'e' [ sign ] digits ] .
sign           = '+' | '-' .
```

例 5.

正しい例
1.0
1.0e-3

正しくない例
.12
e10

3. String Literal

文字列定数を表現するストリングリテラルは，引用符で囲まれた文字の羅列として表現される．

```
string_literal  = \q { character | \s | \o | \q\q } \q .
character       = digit | letter | special .
```

例 6.

```
正しい例
'valid string literal'
'valid string'' literal'

正しくない例
'invalid string' literal'
```

4. Logical Literal

論理定数を表現する論理リテラルは，TRUE，FALSE で表現される．

```
logical_literal = TRUE | FALSE .
```

5. Complexed Literal

LIST, ARRAY などに含まれる値をまとめて表現するために用いられる．

```
complexed_literal = '(' expression
                        { ',' expression } ')'.
```

データタイプ

データタイプには，基本的なデータタイプを定義するシンプルタイプ，配列とリストの要素として用いることができるbase_type ，通常の変数のタイプとして定義できる usual_type とに分類される．

```
usual_types  = base_type  | process_id_type .
base_type    = simple_type | class_type | defined_type |
               array_type | list_type .
simple_type  = integer_type | real_type |
               char_type | string_type
```

1. Integer Type

整数型を定義する integer type の宣言は，予約語 INTEGER で行う．

```
integer_type  = INTEGER .
```

例 7.

```
FUNCTION f1( tmp1 : INTEGER ): INTEGER ;
.....
END_FUNCTION ;
```

2. Real Type

実数型を定義する real type の宣言は，予約語 REAL で行う．

```
real_type   = REAL .
```

例 8.
```
FUNCTION f1( tmp1 : REAL ): REAL ;
.....
END_FUNCTION ;
```

3. Char Type

文字型を定義する char type の宣言は，予約語 CHAR で行う．

例 9.
```
char_type   = CHAR .
```

例 10.
```
FUNCTION f1( tmp1 : CHAR ): CHAR ;
.....
END_FUNCTION ;
```

4. String Type

文字列型を定義する string type の宣言は，予約語 STRING で行う．

```
string_type  = STRING .
```

例 11.
```
FUNCTION f1( tmp1 : STRING ): STRING ;
.....
END_FUNCTION ;
```

5. Logical Type

論理型を定義する logical type の宣言は，予約語 LOGICAL で行う．

```
logical_type  = LOGICAL .
```

例 12.
```
FUNCTION f1( tmp1 : LOGICAL ): LOGICAL ;
.....
END_FUNCTION ;
```

6. Array Type

配列型は，上下限を定義するインデクススペックとベースタイプから定義される．インデクスが一つの場合は，要素の個数を定義し，要素は 0 番から順に用意される．インデクスの定義が 2 項の場合は，配列の上下限を定義しているとし，個数は上限－下限＋ 1 となる．

```
array_type   = ARRAY index_spec OF base_type .
index_spec   = '[' limit [ , limit ] ']' .
limit        = simple_expression .
```

例 13. 始めの例は、1 次元で要素数を 5 としてのみ定義している．従って要素は，0 から 4 まで用意されることになる．次の例では，要素の番地を 1 から 5 までとし，計 6 個の要素を指定している．

```
LOCAL
 11 : ARRAY [5] OF INTEGER ;
 12 : ARRAY [1,5] OF INTEGER ;
```

例 14. 始めの例は，array の array として 2 次元の配列を定義している．次に 3 次元の配列宣言を行っている．

```
LOCAL
 11 : ARRAY [5] OF ARRAY [5] OF INTEGER ;
 12 : ARRAY [5] OF ARRAY [5]
        OF ARRAY [5] OF INTEGER ;
```

7. List Type

リスト型は，要素の個数を定義するインデクススペックとベースタイプから定義される．インデクスがある場合は，要素の個数を定義し，要素は0番から順に格納される．インデクススペックが定義されていない場合は，そのリストは不定長となる．

```
List_type   = LIST [ index_spec ] OF base_type .
index_spec  = '[' limit ']' .
limit       = simple_expression .
```

例 15.
```
LOCAL
 l1 : LIST OF INTEGER ;
 l2 : LIST [5] OF INTEGER ;
```

8. Class Type

ある要素の型としてクラスタイプを宣言する場合は，そのクラス名を用いて行う．

```
class_type   = qualified_id .
```

例 16. ここで，human は別の部分で class として宣言されているという仮定のもとで，局所変数を human クラスのインスタンスとして宣言している．
```
LOCAL
 l1 : human ;
```

9. Defined Type

ある要素の型としてデファインドタイプを宣言する場合は，そのタイプ名を用いて行う．

```
defined_type   = def_type_id .
```

例 17. ここで，length_meger は別の部分でデファインドタイプとして宣言されているという仮定のもとで，局所変数を length_meger タイプのインスタンスとして宣言している．
```
LOCAL
 l1 : length_meger ;
```

10. Process ID Type

ある要素の型としてプロセスタイプを宣言する場合は，そのタイプ名を用いて行う．

```
process_id_type  = PROCESS .
```

例 18. 局所変数 l1 はプロセス型変数として宣言される．
```
LOCAL
 p1 : PROCESS ;
```

11. Pointer Type

ある要素の型として，他のタイプに対するポインタタイプとして宣言する場合は，そのタイプ名の前にハットを用いて行う．

```
pointer_type = '^' base_type .
```

例 19. 局所変数 l1 は整数型に対するポインタとして宣言される．
```
LOCAL
 l1 : ^INTEGER ;
```

言語の基本構造

言語の基本構造構造，値の受け渡し，記述の有効範囲

言語の基本構造

EPDL は基本的にはブロックの一ないしそれ以上の並びとして定義される.

```
EPDL    =  { block } .
block   =  constant_decl_block |
           class_block |
           function_block |
           process_interface_block |
           interface_declaration_block |
           process_action_block |
           type_define_block .
```

例 20.

```
CONSTANT  -- 定数宣言ブロック
  buffer_size : INTEGER := 256 ;
END_CONST ;

CLASS c1; -- クラスブロック
...
END_CLASS

FUNCTION func_example( ) LOGICAL ;
        -- ファンクションブロック

END_FUNCTION ;

INTERFACE i1; -- インターフェースブロック

END_INTERFACE;

PROCESS p1; -- プロセスインターフェースブロック

END_PROCESS;

p1:interrupt(); -- プロセスアクションブロック
BEGIN

END;

TYPE  length_measure := REAL ; -- タイプブロック
```

といったブロックの集合として記述される.

値の受け渡し

プロセス，関数などに値を渡す場合は，Call by Value がデフォルトで実行される. Call by Address を指定する場合は，ポインタタイプ宣言された変数を使用するか，ポインタタイプを返す式を用いる.

アドレスを得るには，変数の前にアドレス演算子を置く.

例 21.

```
i    :  INTEGER ; -- 整数型変数宣言
int_pointer : ^INTEGER ; -- 整数型に対するポインタ宣言
```

```
 :
int_pointer := &i ;
 :
r := event(int_pointer);
r := event(&i);    -- この２つは等価
```

記述の有効範囲

各ブロックの外側で宣言された変数，定数，プロセス等は大域変数として宣言されたものと見なして，続く全てのブロックからアクセスすることができる．

プロセス内及び関数内で宣言された局所変数及び引き数は，そのプロセス又は関数からしかアクセスできない．

他のプロセスの内部変数にアクセスする場合は，相手のプロセスにおいて定義されたメッセージを相手のプロセスに送ることで行う．メッセージを受け取った相手のプロセスは，メッセージに対して定義されたアクションを取る．この場合，その内部変数の値をもとの相手に渡すアクションかもしれないし，内部変数を変更するアクションかもしれない．

記述ブロック

1. Class Block

   ある要素のデータ型として使用されるクラスを宣言するにはクラスブロックを用いて行う．サブタイプ記述及びメンバー記述はオプショナルである．

```
class_block  = CLASS class_id ';'
               [ sub_class_declaration ]
               [ member_declaration ]
               END_CLASS ';' .
subtype_declaration
             = SUB_CLASS OF '(' class_id
                    { , class_id } ')' ';' .
member_declaration
             = member_list ':' usual_types ; .
member_list  = member_id { , member_id } .
member_id    = simple_id .
class_id     = simple_id .
```

例 **22.**

```
CLASS c1;
 mam1 : INTEGER ;
END_CLASS

CLASS c2;
SUB_CLASS OF ( c1 );
 mam1 : INTEGER ;
END_CLASS
```

2. Constant Declaration

   定数を宣言するには，constant_decl_block を用いて行う．定数宣言されたプロセスは，常に存在しているプロセスとして扱われる．

```
constant_decl_block = CONSTANT
```

```
                    { constant_body }
                    END_CONST ';' .

constant_body       = const_id : const_type ':='
                    expression ';' .
const_type          = usual_types .
const_id            = simple_id .
```

例 **23.**

```
CONSTANT
  buffer_size : INTEGER := 256 ;
END_CONST ;
```

3. Defined Type Block

あるタイプを，意味的に重要な別のタイプとして新たに定義するためには
defined_type_block を用いて行う．

```
type_define = TYPE def_type_id ':=' base_type ';'
```

例 **24.**

```
TYPE length_measure := REAL ;
```

4. Process Interface Decralation Block

プロセスインターフェースは，EPDL が記述する世界における全ての情報入出力活動の定義に用いられる．インターフェース定義は，そのプロセスが持つ内部変数，プロセスが受信することができるメッセージ，及びその際に与えられる引き数の定義によって行われる．該当するメッセージを受信した後，プロセスが取る個別のアクションの定義は，インターフェース定義の外で，プロセスアクション定義ブロックによって行われる．

```
process_interface_block  =
                    PROCESS process_id ';'
                  [ inheritance_description ]
                  [ LOCAL
                    local_param_declaration
                    END_LOCAL ';']
                  [ MESSAGE
                    message_decl
                    END_MESSAGE ';' ]
                    END_PROCESS ';'

process_id     = simple_id .

inhericance_description
               = INHERIT_FROM inherit_list ';'.

inherit_list   = process_id { ',' process_id } .

local_param_declaration
               = lp_id { , lp_id } ':' lp_type
                 [ ':=' expression ] ';'.
lp_id          = simple_id .
lp_type        = usual_types .

message_decl
               = [ BROAD_CAST ] message_id
```

```
                formal_parameter_list
                [':' priority_value ]
                .
formal_parameter_list
              = '(' [ fpl_declaration ]
                  { ';' fpl_declaration } ')' .
fpl_declaration
              = fp_id { , fp_id } ':' fp_type .
fp_id         = simple_id .
fp_type       = usual_types .
priority_value
      = integer_literal .
```

例 25. ここではprocess_in_robot と名付けられたプロセスのインターフェースが定義されている．このインターフェースは，局所変数として，INTEGER 型の tmp をもっており，受信可能なメッセージとして，pirtt1 が宣言されている．process_in_robot が pirtt1 を受信した場合のアクションは，別の部分でアクション定義ブロックを用いて宣言される．

```
PROCESS process_in_robot ;
  LOCAL
    tmp : INTEGER ;
  END_LOCAL ;
  MESSAGE
    pirtt1( b1,b2 : bit_type ; d1 : byte_type );
  END_MESSAGE ;
END_PROCESS ;
```

5. Interface Decralation Block

インターフェースは，EPDL が記述する世界における全ての情報の通信対象の定義に用いられる．設計活動者，工作機械，ロボットなどがプロセスの動作に関与するが，これらの通信対象も，プロセスから見た，その通信対象に一対一に対応するプロセスのインターフェースとして宣言される．インターフェース定義は，そのインターフェースが持つ内部変数，インターフェースが受信することができるメッセージ，及びその際に与えられる引き数の定義によって行われる．該当するメッセージを受信した後，インターフェースの反応を表現する場合は，インターフェース定義の外で，プロセスアクション定義ブロックによって行われる．

```
interface_declaration_block  =
                    INTERFACE process_id ';'
                  [ inheritance_description ]
                  [ LOCAL
                    local_param_declaration
                    END_LOCAL ';']
                  [ MESSAGE
                    message_decl
                    END_MESSAGE ';' ]
                    END_INTERFACE ';'

process_id    = simple_id .

inhericance_description
              = INHERIT_FROM inherit_list ';'.
```

```
inherit_list  = process_id { ',' process_id } .

local_param_declaration
              = lp_id { , lp_id } ':' lp_type
                [ ':=' expression ] ';'.
lp_id         = simple_id .
lp_type       = usual_types .

message_decl
              = [ BROAD_CAST ] message_id
                formal_parameter_list
                [':' priority_value ]
                .
formal_parameter_list
              = '(' [ fpl_declaration ]
                  { ';' fpl_declaration } ')' .
fpl_declaration
              = fp_id { , fp_id } ':' fp_type .
fp_id         = simple_id .
fp_type       = usual_types .
priority_value
      = integer_literal .
```

例 **26.** ここでは，process_in_robot と名付けられたインターフェースが定義されている．このインターフェースは，局所変数として，INTEGER 型の tmp をもっており，受信可能なメッセージとして，pirtt1 が宣言されている．process_in_robot が pirtt1 を受信した場合のアクションは，別の部分でアクション定義ブロックを用いて宣言される．

```
INTERFACE process_in_robot ;
  LOCAL
    tmp : INTEGER ;
  END_LOCAL ;
  MESSAGE
    pirtt1( b1,b2 : bit_type ; d1 : byte_type );
  END_MESSAGE ;
END_INTERFACE ;
```

ブロードキャストメッセージ

メッセージには，ブロードキャスティングメッセージと呼ばれる種類がある．通常メッセージは，個別のプロセスに対して発せられるが，ブロードキャスティングメッセージは，相手を指定することなしに，メッセージは「その」メッセージを受信することが可能な全てのプロセスに対して送られる．

インターフェース定義の継承は，内部変数，メッセージ(一般／ブロードキャスト)に関して行われる，引き数の型定義が同一であれば，受信可能とする．引き数の宣言が継承によって変更されていた場合，は受信は不可能とする．引き数定義が同一でも，とられるアクションが再定義されている場合は，その定義に従ってアクションがとられる．またメッセージの再宣言で，ブロードキャスト指定が削除された場合は，そのプロセスはメッセージを受信することができない．

例 **27.** この例では，ブロードキャストメッセージとクロックインターフェースの定義の例を示している．各メッセージにはプライオリティ値が付けられている．

BROADCASTevery_one_hour(current_time : time):300; と
BROADCAST every_one_minute(current_time : time):200; との2つのメッセージが到着した時には1時間間隔の every_one_hour の方が優先される．

```
PROCESS clock_interface ;
  MESSAGE
    BROADCAST every_one_hour( current_time : time ):300;
    BROADCAST every_one_minute( current_time : time ):200;
    BROADCAST every_one_second( current_time : time ):100;
  END_MESSAGE ;
END_PROCESS ;
```

初期化／終了メッセージ

プロセスを初期化した時点で，そのプロセスに自身に対して送られる初期化メッセージがある．これは，プロセスと同じ名前を持っている．また，そのプロセスを終了させるメッセージとして，HALT をデフォルトで各プロセスが持っている．

プロセスを起動する際，及び HALT メッセージを送る際に，引き数を与える場合には，他のメッセージと同様にインターフェース宣言において定義を行う．初期化，及び HALT メッセージのアクションが定義されていない場合，そのままプロセスが生成される／プロセスが消滅してしまうことになる．

一定時間ごとの動作

EPDL では，言語仕様としてではなくモデルとして一定時間毎のメッセージを受信できるクロックインターフェースを設ける．特定時間毎の動作の表現は，このインターフェース定義を継承して，時間毎ごとのメッセージに対するアクションを再定義することで行う．

例 **28.** この例は，クロックインターフェースの定義を示している．

```
INTERFACE clock_interface ;
  MESSAGE
    every_one_hour( current_time : time );
    every_one_minute( current_time : time );
    every_one_second( current_time : time );
  END_MESSAGE ;
END_INTERFACE ;
```

特定時間における動作

特定時間に起動される動作は，一定時間ごとに受信されるクロックインターフェースの引き数として与えられるメッセージ発信時間から，指定された時間にアクションを実行することで表現する．

例 **29.** この例は，クロックインターフェースからの継承と，一定時間ごと動作の再定義を示している．

```
INTERFACE timer_bomb ;
  INHERIT_FROM ( clock_interface );
  MESSAGE
    every_one_hour( current_time : time );
  END_MESSAGE ;
END_INTERFACE ;

timer_bomb : every_one_year
             ( current_time : time ) ;
BEGIN
  IF ( current_time == X-DAY ) THEN
     ; -- fire
END ;
```

アクションプライオリティ

プロセスがあるアクションを実行している途中で，他のプロセスから別のメッセージを受信することがある．この際に，それまで実行していたアクションを中断して，新しいメッセージに対するアクションを実行するか，新しいメッセージの受信をペンディングにしておいて，実行中のアクションを終了させるかが問題となる．

この問題を解決するために，インターフェース宣言におけるメッセージ定義において，メッセージごとにプライオリティを定義できるものとする．

あるメッセージにたいするアクションを実行している際に，それより高いプライオリティを持ったメッセージが到着した場合，実行中のアクションは，ペンディングとして割り込みがかかり，プライオリティの高いメッセージに対するアクションが実行される．実行途中に，実行中のメッセージよりも低いプライオリティのメッセージが到着した場合は，実行中のアクションが優先される．

実行時アクションよりもプライオリティの低いメッセージは，待ち行列に入り，実行中のアクションが終了した時点で，処理される．待ち行列は，プライオリティの高い順番にストアされる．

プライオリティ値が設定されていないメッセージは，最も低い値を持っているとして処理される．

同値のプライオリティを持ったメッセージの場合は，先に到着して処理されているメッセージが優先的に処理される．

例 **30.** この例は，クロックインターフェースの定義を示している．

```
INTERFACE clock_interface ;
  MESSAGE
    every_one_hour( current_time : time ) : 300 ;
    every_one_minute( current_time : time ) : 200 ;
    every_one_second( current_time : time ) : 100 ;
  END_MESSAGE ;
END_INTERFACE ;
```

6. Process Action Decralation Block

プロセスアクションブロックは，定義されたプロセスインターフェースを，各プロセスが受信した場合に，そのプロセスが取る反応を記述したものとする．

プロセスアクションブロックでも，各メッセージごとの引数の記述を行うことができるが，もしインターフェース宣言における記述とアクションブロックにおける引数の記述が矛盾している場合には，アクションブロックの記述が優先される．これは，継承したメッセージの引数の記述が元のインターフェース記述にない場合に，プロセスアクションブロックでの再宣言を優先させるためである．

初期化アクションは，そのプロセスと同じ名前がついたアクションであり，プロセスが生成される時に実行される．終了時アクション HALT は，プロセスを終了消滅させるために送られるメッセージである．プロセスがこのメッセージを受信した時，終了時アクション HALT が定義されていたら，それが実行される．

```
process_block  =
        process_id ':' message_id formal_parameter_list ';'
      [ LOCAL
        local_param_declaration
        END_LOCAL ';']
       BEGIN
       { statement }
       END ';'

process_id    = simple_id .
message_id    = simple_id .

formal_parameter_list
              = '(' [ fpl_declaration ]
                { ';' fpl_declaration } ')' .
fpl_declaration
              = fp_id { , fp_id } ':' fp_type .
fp_id         = simple_id .
fp_type       = usual_types .

local_param_declaration
              = lp_id { , lp_id } ':' lp_type
                [ ':=' expression ] ';' .
lp_id         = simple_id .

lp_type       = usual_types .
```

例 **31.**

```
INTERFACE process_robot ;
  MESSAGE
    move_to_home_position( );
        :
  END_MESSAGE ;
END_INTERFACE ;

process_in_robot : move_to_home_position( ) ;
LOCAL
 tmp : INTEGER ;
END_LOCAL ;
BEGIN
   ....
END ;
```

7. Function Block Decralation

ファンクションは，通常のプログラミングにおける関数呼び出しと同じ様に，呼び出し側の実行が一事的に中断されて，呼びだした関数側に移る形を取る．その再のプロセスＩＤは，呼び出し側のものとは異なった値を取る．

```
function_block  =
        FUNCTION function_id formal_parameter_list
            return_type ';'
        [ LOCAL
          local_param_declaration
          END_LOCAL ';']
        { statement }
        END_FUNCTION ';'

function_id    = simple_id .
formal_parameter_list
               = '(' [ fpl_declaration ]
                 { ';' fpl_declaration } ')' .
fpl_declaration
               = fp_id { , fp_id } ':' fp_type .
fp_id          = simple_id .
fp_type        = usual_types .
return_type    = usual_types .

local_param_declaration
               = lp_id { , lp_id } ':' lp_type
                  [ ':=' expression ] ';' .
lp_id          = simple_id .

lp_type        = usual_types .
```

例 **32.**

```
FUNCTION func_example( ) LOGICAL ;
LOCAL
 tmp : INTEGER ;
END_LOCAL ;

....

END_FUNCTION ;
```

実行文

プロセスやファンクションブロックにおいて，実際にアルゴリズムを記述するために出現するのが statement である．

```
statement = assignment_statement | delay_statement |
            repeat_statement | if_statement |
            compound_statement | retuen_statement |
            while_statement | escpe_statement |
            skip_statement | null_statement |
            transaction_call_statement | within_statement |
            transaction_accept_statement .
```

1. 代 入 文

assignment_statement は，代入演算を行う文である．代入演算の結果そのものは値としては返されない．[1]

```
assignment_statement = general_ref ':=' expression ';' .
general_ref          = ( fp_id | lp_id | member_id )
                         { qualifier } .
qualifier            = member_qualifier | index_qualifier .
member_qualifier     = '.' member_id .
index_qualifier      = '[' expression
                         { , simple_expression } ']'.
```

2. 遅延動作文

一定時間プロセスの実行を休止することを指定する文．単位は秒で与えられる．

```
delay_statement  = DELAY '(' expression ')' ';' .
```

3. 繰り返し文

```
repeat_statement  = REPEAT increment_control
                    { statement }

increment_control = general_ref ':=' bound_1 TO bound_2
                      [ BY increment ].
bound_1           = simple_expression .
bound_2           = simple_expression .
increment         = simple_expression .
```

例 **33.**

```
REPEAT i := 1 TO 5
 a := a + 1 ;

REPEAT i := 1 TO 5 BY 2
BEGIN
 a := a + 1 ;
END ;
```

4. If 文

If 文は，IF と THEN の間の expression を評価し，TRUE が返された場合は，THEN 以降の文を実行する．ELSE とそれ以降の実行文が用意されており，式が FALSE と評価された場合は，ELSE 以降の文が実行される．式が TRUE/FALSE 以外の値を返した場合には，どの文も実行されない．複数の文を実行する場合には，コンパウンド文を用いる．

```
if_statement  = IF expression THEN statement
                  [ ELSE statement ]
```

例 **34.**

```
if a =< 100 THEN a := a + 1 ;
if a =< 100 THEN a := a + 1 ; ELSE a:= 0;

if a =< 100 THEN
BEGIN
```

[1] array や list タイプとして宣言された識別子の要素にアクセスする場合は，integer_array[5] のように指定して内部要素の値にアクセスする．

```
END ;

if a =< 100 THEN
BEGIN

END ;
ELSE BEGIN

END ;
```

5. While 文

`while_statement` は，expression の評価値が TRUE の間，statement を実行して再び expression を評価する．常に TRUE を返す数式であれば無限ループとなる．

```
while_statement  = WHILE expression DO
                   statement .
```

例 **35.**

```
WHILE x < 100 DO
 x := x + 1 ;

WHILE x < 100 DO
BEGIN
 x := x + 1 ;
END ;
```

6. ブロック文

`compound_statement` は，if 文や while 文などで複数の文を実行するために用いられる．

```
compound_statement = BEGIN
                     { statement }
                     END ';'
```

7. リターン文

function 中で結果を返す場合に用いられる．実行は，呼ばれた時点で終了して，式の値が評価されて関数の結果として返される．

```
return_statement = RETURN expression ';' .
```

例 **36.** 引数として ii，返り値の型として LOGICAL を指定．計算アルゴリズムの後で `return_value` の値が決定され，関数の値として返される．

```
FUNCTION example_function( ii:int ):LOGICAL ;
LOCAL return_value : LOGICAL ;
(* something algorythm *)
    RETURN( return_value );
END_FUNCTION ;
```

8. スキップ文

スキップ文は，その文が存在するコンパウンド文の終端に実行を写す．

```
skip_statement   = SKIP ';' .
```

例 **37.**

```
WHILE x < 100 DO
BEGIN
  x := x + 1 ;
  ....
IF    THEN SKIP ; -- SKIP が呼ばれると実行は END
                  -- まで進み, WHILE 文の論理条件
                  -- 式が評価される.
  ....
 END ;
```

9. エスケープ文

エスケープ文は，その文が存在するコンパウンド文から抜け出す.

```
escpe_statement  = ESCAPE ';' .
```

例 **38.**

```
WHILE x < 100 DO
BEGIN
 x := x + 1 ;
 ....
 IF    THEN ESCAPE ; -- ESCAPE が呼ばれると実行は
                     -- END の後ろまで進み, WHILE 文
                     -- のくり返しを抜け出す.
 ....
END ;
```

10. Null 文

null_statement は，何もしない文で，if 文や repeat 文などで用いられる.

```
null_statement  = ';'
```

数式

数式 (Expression) は，各計算式等で用いられる一般的な数式の書式を定義する.

```
expression     = simple_expression
                   [ rel_op simple_expression ] .
rel_op         = '<' | '>' | '<=' | '>=' | '<>' | '=' .
simple_expression
               = term { add_like_op term } .
term           = factor { mult_like_op factor } .
factor         = simple_factor [ '*' simple_factor ] .
simple_factor = literal | initializer | qualifiable_factor |
                '( expression ')' | unary_op simple_factor |
                address_op simple_factor | function_call |
                process_start | message_sending .

qualifiable_factor
               = general_ref { qualifier } .
unary_op       = '-' | '+' | NOT .
mult_like_op   = '*' | '/' | DIV | MOD | AND .
add_like_op    = '+' | '-' | OR | XOR .
address_op     = '&' .
```

プロセス起動

プロセスの起動は以下の構文で行う．この式は開始されたプロセスＩＤを返す．代入式中ではこの値はプロセスタイプとして宣言された変数に代入されなければならない．

```
process_start    = ( buitlein_process_id | process_id )
                   '(' actual_parameter_list ')' ';' .
actual_parameter_list
                 =  parameter { ',' parameter } .
parameter        = expression .
```

例 39. ここで，chield_process は，プロセスタイプの変数を一つを引き数として起動されるプロセスとして定義されているものと仮定する．この例では，起動する親のプロセスが自分のプロセスＩＤを得た後で，これを引き数として与えている．

```
LOCAL
 self, P1 : PROCESS ;
BEGIN
 self := SELF_PROCESS();
 P1 := chield_process( self );
```

メッセージ発信

他のプロセスへのメッセージの発信は以下の構文で行う．この式はメッセージが相手に受信されたかどうかの結果を返す．代入式中ではこの値は論理型として宣言された変数に代入されなければならない．

```
message_sending = ( process_id_list |
                    buitlein_process_id | process_id ) .
                  message_id '(' actual_parameter_list ')' ';' .
actual_parameter_list
                =  parameter { ',' parameter } .
process_id_list =  '(' process_id { ',' process_id } ')'
parameter       = expression .
```

例 40. この例では，起動する親のプロセスが自分のプロセスＩＤを得た後で，これを引き数として与えている．次ぎにプロセス P1 に対して P1 を引き数としたメッセージ child_process_message が送られる．

```
LOCAL
 self, P1, P2 : PROCESS ;
 tmp : INTEGER ;
 result : LOGICAL ;
BEGIN
 self := SELF_PROCESS();
 result := P1.child_process_message( tmp );
 result := ( P1, P2 ).child_process_message( tmp );
```

関数呼び出し

```
function_call  = ( buitlein_function_id | function_id )
                   '(' actual_parameter_list ')'  ';' .
actual_parameter_list
               = parameter { ',' parameter } .
parameter    = expression .
```

例 41. この例では，論理型の変数を宣言し，次にプロセスＩＤタイプを引き数として，そのプロセスが活動中か否かを代えす関数を呼んでいる．

```
LOCAL
 flag : LOGICAL ;
BEGIN
 flag := IS_ACTIVE ( chiled_process );
```

プロセス操作関数

組み込み関数として現在実行されている自分自身のプロセスＩＤを返す．

- process_id_type SELF_PROCESS()

プロセス ID を与えることで，そのプロセスと，そのプロセスのチャイルドプロセスを中止させる関数を定義する．

- int HALT(process_id_type)

プロセス ID を与えられることで，そのプロセスの状態を返す．正しければ TRUE を返し，満たしていなければ FALSE を返す

- IS_COMPLETED(process_id_type) : LOGICAL そのプロセスが完了しているか
- IS_ACTIVE(process_id_type) : LOGICAL そのプロセスが活動中か

モデルの記述

本言語では，記述の対象とされる活動を情報の入出力の現象として，又活動に関与する対象を情報の入出力対象として，その活動の中で入出力される情報の記述を目的としている．

本言語は，基本的には上記の活動／入出力対象の構造を定義する．具体的な活動あるいは対象を記述する際には，それらの活動／対象を定数として記述する．

特定の活動が動作している間に，新たな情報の入出力対象が出現する際には，活動中のプロセスに入出力対象に対して，新たな入出力対象からメッセージが発せられる．活動中のプロセスは，新たな入出力対象の識別子を変数として格納する必要がある．何等かの形で，プロセス中の変数として確保できない場合は，そのプロセスから新しい入出力対象に対してメッセージを発信することはできない．

B　付 録：設計グループ活動モデルの EPDL による記述

基本構造

このモデルは，設計グループの各設計者及び管理者の間での情報の交換及び，設計グループのメンバーから設計データに対するアクセスを制限するためのプロセスのモデル化を行っている．

現行の EPDL の言語仕様では，プロセスのイベント処理の排他的制御を行うことができないため，特定のメッセージ処理を行っている間に他のプロセスにおけるメッセージ処理を制限することはできず，各プロセスは完全に等価に並行して動作する．このために，あるプロセスに対して別のプロセスから何等かの反応が帰って来たとき，何時の時点の内部状態に対しての反応かを特定することが出来ない．

この問題を解決するために，あるイベントを起こす際に付帯した状況に関する情報 (イベントの発信者，発信したプロセス，対象となっている設計データ) などの情報をまとめて一つのデータとして，常に入出力の際に付加することとした．これによって設計活動プロセスは，ある処理を行った際の状況を全て，メッセージに付加して送り出し，またそれを受け取ることで，どの要求に対する返答が帰って来たかを把握することが可能になる．

```
*)
-------------------------------------------------------------------------------
-- Concurrent Design Process Model Rev 2.0
-- Edited by Yoshihito KIKUCHI
-------------------------------------------------------------------------------
(*
```

型定義

ここでは，本モデルで用いられるデータタイプの定義を行っている．便宜定義しただけで，実体は存在しないデータタイプもある．より具体的な作業モデルを構築する際には，これらのタイプは厳密に定義される必要がある．

```
*)
-------------------------------------------------------------------------------
-- Type Definition
-------------------------------------------------------------------------------
-- プロセスが活動中か否かを決めるためのフラグのタイプ
TYPE process_mode := INTEGER;

-- CAx で用いられる一般的なデータのためのクラス
CLASS generic_data ;
    --
END_CLASS ;

-- 設計者を識別するためのクラス
CLASS personal_id ;
    --
END_CLASS ;

-- 設計データを識別するためのクラス
CLASS design_data_id ;
    --
END_CLASS ;

-- 設計プロセスのスケジュールデータのためのクラス
```

```
CLASS design_schedule_data ;
    limit_date : date ;
END_CLASS ;

-- 設計グループを定義するためのクラス
CLASS design_group_data ;
    administrator      : personal_id ;
    designers          : LIST of personal_id;
    inner_design_group : LIST of design_group_data ;
    refer_enable_datas : LIST OF generic_data ;
    changeable_datas   : LIST OF generic_data ;
END_CLASS;

-- 設計プロセスを定義するためのクラス
CLASS design_process_info ;
    group_data : design_group_data ; -- process interface for group
    schedule   : design_schedule_data ;          -- process schedule data
END_CLASS ;

-- 設計者から設計プロスに対するイベントンの引き数として渡されるための
-- 汎用的なデータ
CLASS designer_command
    designer        : user_interface ; -- owner of this designer_command
    source          : PROCESS ; -- process that is correspond
    data_ids        : LIST OF design_data_id ;
    datas           : LIST OF generic_data ;
    process_list    : LIST OF design_process_info ;
END_CLASS ;
(*
```

定数定義

ここでは，本モデル中で用いられる定数を宣言している．具体的には，設計活動プロセスの内部状態を表現するための整数値のフラグと，設計グループの活動に関する情報を格納するデータベースへのインターフェースを定数として宣言している．

```
*)
-------------------------------------------------------------------------
-- Constant Value Definition
-------------------------------------------------------------------------
CONSTANT
 SLEEP              : process_mode := 1 ; -- process is sleeping
 WAKE_UP            : process_mode := 2 ; -- process is wakeuped

 -- グループ活動に関するデータベースへのインターフェース
 management_database : design_process_database_interface ;

 -- 製品データベースへのインターフェース
 product_database : product_database_interface ;

END_CONST;
(*
```

インターフェース定義

ここでは，本モデル中で用いられるインターフェースの定義を行っている．具体的には，一定時間毎の設計活動プロセスの動作を表現するためのクロックインターフェース，設計グループの活動に関する情報を格納するデータベースへのインターフェース，個別の設計者に対するインターフェースが定義される．

動的に変化する設計グループの構造は，プロセスとして次節で定義する．

```
*)
------------------------------------------------------------------------
-- Interface Definition for Designer and Database Interface
------------------------------------------------------------------------
-- Database includes security informations about design information reference
-- and change, correspondence of design data and design group, process
-- precedence
-- Database accepts a generic data_base_call designer_command and returns a result with
-- database_reslt designer_command

-- 一定時間毎の動作を定義するためのインターフェース
PROCESS clock_interface ;
  MESSAGE
    BROADCAST every_one_date( current_time : time ):400;
    BROADCAST every_one_hour( current_time : time ):300;
    BROADCAST every_one_minute( current_time : time ):200;
    BROADCAST every_one_second( current_time : time ):100;
  END_MESSAGE ;
END_PROCESS ;

-- 設計プロセスデータを格納したデータベースに対するインターフェース
INTERFACE design_process_database_interface ;
INHERIT_FROM ( data_resource );
  MESSAGE ;
    right_change_request( source : PROCESS ; user_command : designer_command );
    data_refer_request( self: PROCESS; user_command : designer_command);
    data_change_result( result : LOGICAL );
    request_chield_data(self: PROCESS; user_command : designer_command);
    request_next_chield_process(self : PROCESS ; chield : designer_command);
  END_MESSAGE ;
END_INTERFACE ;

-- 設計データを格納したデータベースに対するインターフェース
INTERFACE product_database_interface ;
INHERIT_FROM ( data_resource );
  MESSAGE ;
    data_change_request( self: PROCESS ; user_command : designer_command );
    data_refer_request( self: PROCESS; user_command : designer_command);
  END_MESSAGE ;
END_INTERFACE ;

-- 個々の設計者のためのインターフェース
INTERFACE user_interface ;
INHERIT_FROM ( external_environment );
  MESSAGE
    right_change_announce( user_command : designer_command ; result: LOGICAL );
    send_message( source : PROCESS ; result : LOGICAL );
    approval_call( source : PROCESS ; data : generic_data );
```

```
    schedule_limit( current_time ; time );
    process_end_validate( result : LOGICAL ; chield : PROCESS );
    design_data_approval_request (self : PROCESS ; user_command : designer_command);
    design_data_approval_result(self : PROCESS; user_command ; designer_command);
  END_MESSAGE ;
END_INTERFACE ;

(*
```

プロセス定義

ここでは，本モデル中で用いられる２つのプロセスを宣言している．具体的には，仮想的な設計グループの構造を定義するためのプロセス，設計活動における各種のイベントの処理を行う 動的設計プロセスモデルである．

設計グループ構造を定義するためのプロセス

このプロセスは，動的設計プロセスモデルが起動されると，その作業プロセスが担当する組織構造を表現するために起動される．具体的には，設計活動プロセスに発生したイベントの中で，グループ全体に対して情報の配布を行い，設計作業者からグループ宛で発信した情報を，当該設計活動プロセスに対して送り出す働きを行う．

```
*)
-------------------------------------------------------------------------------
-- Process to interface about virtual design group
-------------------------------------------------------------------------------
-- 仮想的な設計グループ構造を定義するためのプロセス
PROCESS DesignGroupInterface ;
  LOCAL ;
    administrator      : user_interface ;
    designers          : LIST of user_interface;
    inner_design_group : LIST of design_group_data ;
  END_LOCAL ;
  MESSAGE
    -- グループに対する承認要求
    design_data_approval_request
         (source : PROCESS ; user_command : designer_command);

    -- グループに対する承認要求結果の処理
    design_data_approval_result(source : PROCESS ; user_command : designer_command);
  END_MESSAGE;
END_PROCESS;

DesignGroupInterface : DesignGroupInterface ( initial_group_data : design_group_data )
LOCAL
    count : INTEGER ;
    r1 := LOGICAL ;
END_LOCAL;
BEGIN
    administrator := user_interface(personal_id);

    for count := 1 to size( initial_group_data.user_interface ) BEGIN
        r1 := add( designers,
                   user_interface(initial_group_data.user_interface[count]) );
    END;
```

```
    for count := 1 to size( initial_group_data.design_group_data ) BEGIN
        r1 := add( inner_design_group,
                design_group(initial_group_data.inner_design_group[count]) );
    END;
END;

-- 設計データの承認に関する要求を受け取ったら，設計者の一人一人に対して
-- その設計データの承認の要求を出す.
DesignGroupInterface : design_data_approval_request
    (source : PROCESS ; user_command : designer_command)
LOCAL
    count : INTEGER ;
END_LOCAL;
BEGIN
    for count := 1 to size( designers ) BEGIN
       designers[count].design_data_approval_request ( source , user_command);
END;

-- 個別の設計者から承認の結果が発信されたら，承認メッセージを発信した元に対して
-- 結果を返す.
DesignGroupInterface : design_data_approval_result
    (source : user_interface ; user_command : designer_command)
BEGIN
    designer_command.source.design_data_approval_result( source, designer_command );
END;
(*
```

動的設計プロセスモデル

動的設計プロセスモデルは，設計活動中に発生するイベント毎に情報の伝達や伝達の制限を行うためのモデルである．また，一定時間後とに終了期限かどうかのチェックを行い，期限になった場合は，管理者に対してメッセージを発信している．

```
*)
-------------------------------------------------------------------------------
-- Design Group Process Definition
-------------------------------------------------------------------------------
(*
```

動的設計プロセスモデルインターフェース定義

```
*)
-------------------------------------------------------------------------------
-- Design Group Process Interface
-------------------------------------------------------------------------------
PROCESS ActiveDesignGroupProcess
    ( started_from : PROCESS ; info : design_process_info )
INHERIT FROM (clock_interface, GroupActivity) ;
MESSAGE
-- データ参照／変更権利の変更
right_change ( user_command : designer_command );
right_change_result ( result : designer_command : result : LOGICAL);

-- データの変更
data_change ( user_command : designer_command );
data_change_result ( user_command : designer_command result : LOGICAL );
```

```
-- データの参照
data_refer ( user_command : designer_command );
data_refer_result
    ( user_command : designer_command ; result : LOGICAL ; data : generic_data );

-- データの子プロセスの開始／作業の分割
chield_start ( user_command : designer_command );
chield_data ( user_command : designer_command  ; info : design_process_info );

-- 作業プロセスの中断
pose( user_command : designer_command );

-- 作業プロセスの再開
wake_up( user_command : designer_command );

-- 子プロセスの終了
chield_process_ended( chield : PROCESS ; );

-- 子プロセスの強制的な停止
chield_process_halt( user_command : designer_command );
chield_halt_validate( user_command : designer_command   ; result : LOGICAL );

-- 設計作業データの承認
design_data_approval( user_command : designer_command );
design_data_approval_request( source : PROCESS ; user_command : designer_command );
design_data_approval_result(source : user_interface ; user_command : designer_command );

-- 1日毎のスケジュール確認
BROADCAST every_one_date( current_time : time ):400;

LOCAL
    chileds : LIST OF PROCESS;
    mode    : process_mode := WAKE_UP ; -- self process is wakeuped
    self, parent, group : PROCESS;
    self_info : design_process_info ;
    r1 : LOGICAL ;
    r2 : INTEGER;
END_LOCAL;
END_PROCESS ;
(*
```

動的設計プロセスモデルイベント対応定義

```
*)
-------------------------------------------------------------------------------
-- Design Group Process Event Definition
-------------------------------------------------------------------------------
(*
```

設計活動プロセスの初期化

設計活動プロセスは，初期化時に自身に間する情報を与えられる．この情報に従って管理対象となる設計グループへの情報の配分を行うプロセスを開始する．

```
*)
-------------------------------------------------------------------------------
```

```
-- process initialize
--------------------------------------------------------------------------------
ActiveDesignGroupProcess : ActiveDesignGroupProcess(
    started_from : PROCESS ;
    info : design_process_info )
BEGIN
    parent := started_from ;
    self_info := info ;
    self := SELF_PROCESS(); -- get self process id
    group := DesignGroupInterface ( info.design_group_data );
    -- start virtual design group
END;
(*
```

管理者からの参照／変更権利の変更

各設計グループは，参照可能／変更可能な設計データの ID をあらかじめ規定されており，設計データの変更や参照の要求が起こった場合には，そのデータを参照することで，設計作業者に対して設計データを開示するかどうかの判断を行う．

そのための参照／変更の権利は，グループの管理者から再指定することが可能であり，このイベントは，その際の設計活動プロセスの動きを定義している．

```
*)
--------------------------------------------------------------------------------
-- request to change reference right of data base , called by group manager
--------------------------------------------------------------------------------
ActiveDesignGroupProcess : right_change ( user_command : designer_command ; )
BEGIN
    -- コマンドの発信者が当該プロセスの管理者の場合に
    IF ( mode == WAKE_UP AND
         user_command.designer == design_process_info.group_data.administrator)
    THEN BEGIN
        r1 := management_database.right_change_request( self, user_command );
    END;
END;

ActiveDesignGroupProcess : right_change_result
     (user_command : designer_command : result : LOGICAL)
BEGIN
     -- 処理したコマンドと，その結果をコマンド発信者に返す
     designer_command.designer.right_change_announce( user_command, result );
END;
(*
```

設計データの変更要求

設計者から設計データを添えて，データの変更が要求された際のプロセスの対応．

```
*)
--------------------------------------------------------------------------------
-- request to data change, called by generic designer
--------------------------------------------------------------------------------
ActiveDesignGroupProcess : data_change ( user_command : designer_command )
LOCAL
    num : INTEGER ;
BEGIN
```

```
    -- コマンドの発信者が当該プロセスの設計者グループに入っているか
    IF (     mode == WAKE_UP
         AND IN(design_process_info.group_data.designers, user_command.designer) )
    THEN BEGIN
         for num := 1 to size( user_command.datas ) BEGIN
             IF ( IN( user_command.datas[num], design_process_info.dataset) )
             THEN BEGIN
                -- 指定されたデータが変更可能なデータの場合
                product_database.data_change_request( self, user_command );
             ENSE BEGIN
                user_command.designer.data_change_result( FALSE );
             END;
        END; -- of for
    END;
END;

-- データベースからの変更結果を，変更要求した設計者に返す
ActiveDesignGroupProcess : data_change_result (
        user_command : designer_command ;
        result : LOGICAL
)
BEGIN
    IF ( mode == WAKE_UP AND result == TRUE) THEN BEGIN
       r1 := user_command.designer.data_change_result( user_command, result );
    END;
END;
(*
```

設計データの参照要求

設計者から設計データのIDを添えて，設計データの参照要求が出された際のプロセスの反応．

```
*)
-------------------------------------------------------------------------------
-- design data reference request
-------------------------------------------------------------------------------
ActiveDesignGroupProcess : data_refer(
        user_command : designer_command      -- user generic designer_command
)
LOCAL
    num : INTEGER ;
BEGIN
    -- コマンドの発信者が当該プロセスの設計者グループに入っている
    IF (     mode == WAKE_UP
         AND IN(design_process_info.group_data.designers, user_command.designer) )
    THEN BEGIN
         for num := 1 to size( user_command.datas ) BEGIN
             IF ( IN( user_command.datas[num], design_process_info.dataset) )
             THEN BEGIN
                -- 指定されたデータが変更可能なデータの場合
                product_database.data_refer_request( self, user_command );
             ENSE BEGIN
                user_command.designer.data_refer_result( FALSE );
             END;
        END; -- of for
    END;
END;
```

```
ActiveDesignGroupProcess : data_refer_result (
        user_command : designer_command ;
        result : LOGICAL ;
)
BEGIN
    IF ( mode == WAKE_UP AND result == TRUE) THEN BEGIN
      r1 := user_command.designer.data_change_result( result , user_command);
    END;
END;
(*
```

設計活動プロセスの中断

グループの管理者から発信することが可能であり，このメッセージを受け取ると設計活動プロセスは，睡眠状態になり，管理者からの再起動以外のメッセージを受け取らない．

```
*)
-------------------------------------------------------------------------------
-- process pose
-------------------------------------------------------------------------------
ActiveDesignGroupProcess : pose(
        user_command : designer_command
)
BEGIN
    IF (mode == WAKE_UP AND
        user_command.designer == design_process_info.group_data.administrator)
    ) THEN BEGIN
       for num := 1 to size( chields ) BEGIN
           -- send pose message to chield
           chields[num].pose(user_command);
        END; -- of for
        mode := SLEEP ;
    END; -- of for
END;
(*
```

設計活動プロセスの再開

設計グループの管理者から再開のメッセージが送られると，設計活動プロセスは他の設計作業者からの作業要求を受け付けることが可能になる．

```
*)
-------------------------------------------------------------------------------
-- process restart
-------------------------------------------------------------------------------
ActiveDesignGroupProcess : wake_up(
        user_command : designer_command
)
BEGIN
    IF (mode == SLEEP AND
        user_command.designer == design_process_info.group_data.administrator)
    )
     THEN BEGIN
        for num := 1 to size( chields )    BEGIN
            -- send wake up message to chield
            chields[num].wake_up(self,user_command);
```

```
        END; -- of for
    END; -- of if
END;
(*
```

子プロセスの開始

子プロセスの開始，即ち設計活動の細分化は，そのグループの管理者から発せられる．メッセージを受け取った設計活動プロセスは，子プロセスの情報をデータベースから受け取り，起動を行い，自身の子プロセスのリストに格納する．

```
*)
-------------------------------------------------------------------------------
-- chield process start designer_command
-- this algprothm does not stated process that is to be wake uped
-- this point should be fixed
-------------------------------------------------------------------------------
ActiveDesignGroupProcess : chield_start (
        user_command : designer_command
)
BEGIN
    -- コマンドの発信者が当該プロセスの管理者の場合に
    IF ( mode == WAKE_UP AND
         user_command.designer == design_process_info.group_data.administrator)
    THEN BEGIN
        -- get process information from data base
        management_database.request_chield_data(self,user_command);
END;

ActiveDesignGroupProcess : chield_data (
        user_command : designer_command  ;
        info : design_process_info
)
BEGIN
    -- get process information from data base
    -- add process to process list
    r1 := add( chields, ActiveDesignGroupProcess (self, info));
END;
(*
```

子プロセスの終了

作業が完了して子プロセスが終了する際のイベント

```
*)
-------------------------------------------------------------------------------
-- chield process is completed and stopped
-------------------------------------------------------------------------------
ActiveDesignGroupProcess : chield_process_ended(
        chield : PROCESS ;
)
BEGIN
    IF (mode == WAKE_UP) THEN BEGIN
        -- グループ管理者に当該プロセスをストップさせるかを確認
        design_process_info.group_data.administrator.
            process_end_validate(self, chield);
    END;
```

```
END;

ActiveDesignGroupProcess : process_end_validate(
        result : LOGICAL ;
        chield : PROCESS
)
BEGIN
    IF (mode == WAKE_UP AND result == TRUE) THEN BEGIN
        -- 子プロセス全部と設計グループインターフェースと自身をストップさせる
        parent.chield_process_ended(self);
        group.halt();
        self.halt();
    ELSE BEGIN
        -- 当該プロセスをストップさせずに，次の子プロセスを開始する.
        management_database.request_next_chield_process(self, chield);
    END;
END;
(*
```

子プロセスの強制的な停止

このイベントは，当該設計グループの管理者から発せられる．プロセスは，発信者が管理者であることを確認してから，子プロセスの設計作業プロセスを消滅させる．

```
*)
-------------------------------------------------------------------------------
-- request to stop the cheild process
-------------------------------------------------------------------------------
ActiveDesignGroupProcess : chield_process_halt(
        user_command : designer_command
)
BEGIN
    IF ( mode == WAKE_UP AND
         user_command.designer == design_process_info.group_data.administrator)
    ) THEN BEGIN
        for num := 1 to size( user_command.process_list )    BEGIN
            -- 与えられた process_list に halt を送る.
            R2:=user_command.process_list[num].halt();
            REMOVE( chield , process_list[num] );
        END; -- of for
    END; -- of if
END;
(*
```

設計データの承認

設計作業者から設計活動プロセスに対して承認の要求を出すと，設計作業プロセスは正規の作業者であることを確認して，管理者が承認権を持っている場合は管理者に，同じ設計グループの設計作業者も承認権 (変更権) を持っている場合は，それぞれの設計作業者に対して承認要求のメッセージを送信する．さらに，上位のプロセスに対しても承認の要求を発信する．

帰って来た結果は，当該プロセスが担当している作業者が発した承認要求であれば，その設計作業者に返され，別のプロセス (多分子プロセス) からの承認要求の場合は，該当するプロセスに対して承認の結果が返される．

```
*)
```

```
--------------------------------------------------------------------------------
-- design data apprval request from chield process or opertor
--------------------------------------------------------------------------------
-- プロセスが最初に受ける要求
ActiveDesignGroupProcess : design_data_approval(
        user_command : designer_command
)
BEGIN
    -- コマンドの発信者が当該プロセスの設計者グループに入っている
    IF ( mode == WAKE_UP AND
         AND IN(design_process_info.group_data.designers, user_command.designer) )
    ) THEN BEGIN
        -- 承認要求のデータ一つ一つに対して
        for num := 1 to size( user_command.datas ) BEGIN
            IF ( IN( user_command.datas[num],
                     design_process_info.group_data.refer_enable_datas) )
            THEN BEGIN
            -- 指定されたデータが参照可能なデータの場合
            -- 管理者は変更可能な場合
               parent.design_data_approval_request(self, user_command);
                                        -- 親プロセスに対する承認要求
               self_info.group_data.design_data_approval_request
                    (self, user_command);-- グループ管理者に対する承認要求

            -- 指定されたデータが変更可能な場合
            ENSE IF ( IN( user_command.datas[num],
                      design_process_info.group_data.changeable_datas) )
            THEN BEGIN
               group.design_data_approval_request
                    (self, user_command);-- グループに対する承認要求
                                        -- 管理者に対しては別
            ELSE BEGIN
               user_command.designer.design_data_approval_result( FALSE );
            END;
        END; -- of for
    END;
    END ;
    -- 当該データの承認要求を行うべきならグループと親プロセスに要求
END;

-- 承認回答の受け取り
ActiveDesignGroupProcess : design_data_approval_result(
        source : user_interface ;
        user_command : designer_command
)
BEGIN
    -- 自分のグループのデータなら要求を出した設計者に返す
    IF ( mode == WAKE_UP AND source == designer_command.designer) THEN BEGIN
        designer_command.designer.design_data_approval_result(self, user_command);
    ELSE BEGIN
    -- 子プロセスのデータなら要求を出したプロセスに返す
        designer_command.source.design_data_approval_result(self, user_command);
    END ;
END;

ActiveDesignGroupProcess : BROADCAST every_one_date( current_time : time ):400;
```

```
BEGIN
    IF (current_time == limit_date ) THEN BEGIN
        info.group_data.administrator.schedule_limit( current_time );
    END;
END;
```

C　付 録：生産活動モデルの EPDL による記述

基本構造

このモデルは，生産現場における生産／搬送活動を表現するために，データベースから生産設備に対する情報の出力，搬送設備に対する情報の出力を行うプロセスのモデル化を行う．

型定義

ここでは，本モデルで用いられるデータタイプの定義を行っている．便宜定義しただけで，実体は存在しないデータタイプもある．より具体的な作業モデルを構築する際には，これらのタイプは厳密に定義される必要がある．

```
*)
-------------------------------------------------------------------------------
-- Concurrent Manufacturing Process Model Rev 1.0
-- Edited by Yoshihito KIKUCHI
-------------------------------------------------------------------------------

-------------------------------------------------------------------------------
-- Type Definition
-------------------------------------------------------------------------------
-- 生産スケジュール中で指定するためのフラグ
TYPE event_num : INTEGER ;

-- CAx で用いられる一般的なデータのためのクラス
CLASS generic_data ;
   --
END_CLASS ;

-- 生産設備を識別するためのクラス
CLASS machine_tool_id ;
   --
END_CLASS ;

-- 生産データを識別するためのクラス
CLASS manufacturing_data_id ;
   --
END_CLASS ;

-- 生産プロセスのスケジュールデータのためのクラス
CLASS manufacturing_schedule_data ;
   start_time : time ;
   event : event_num ;
   machine_tool : machine_tool_interface ; -- interface target machine tool
   data_id : manfacturing_data_id ; -- datas to be send
   chield_process_info : LIST of man_process_info ; -- to start chield process
END_CLASS ;

-- 生産プロセスを定義するためのクラス
CLASS man_process_info ;
   schedule     : LIST of manufacturing_schedule_data ;  -- process schedule data
END_CLASS ;


(*
```

定数定義

ここでは，本モデル中で用いられる定数を宣言している．具体的には，生産プロセスが参照するスケジュール内部で指定されるイベントの識別フラグと設計グループの活動に関する情報を格納するデータベースへのインターフェースを定数として宣言している．

```
*)
-------------------------------------------------------------------------------
-- Constant Value Definition
-------------------------------------------------------------------------------
CONSTANT
 CHIELD_START   : event_num := 1 ;      -- 子プロセスの開始
 DATA_TRANSFER : event_num := 2 ;      -- 対象マシンに対するデータの送信

 -- 生産活動に関するデータベースへのインターフェース
 scedule_database : manufacturing_schedul_database_interface ;

 -- 生産情報を蓄えるデータベースインターフェース
 machining_database : manufacturing_database_interface ;

END_CONST;
(*
```

インターフェース定義

ここでは，本モデル中で用いられるインターフェースの定義を行っている．具体的には，一定時間毎の設計活動プロセスの動作を表現するためのクロックインターフェース，設計グループの活動に関する情報を格納するデータベースへのインターフェース，個別の生産機械／搬送機械に対するインターフェースが定義される．

```
*)
-------------------------------------------------------------------------------
-- Interface Definition for Manufacturin Database Interface
-------------------------------------------------------------------------------

-- 一定時間毎のメッセージを受信するためのインターフェース
PROCESS clock_interface ;
  MESSAGE
    BROADCAST every_one_date( current_time : time ):400;
    BROADCAST every_one_hour( current_time : time ):300;
    BROADCAST every_one_minute( current_time : time ):200;
    BROADCAST every_one_second( current_time : time ):100;
  END_MESSAGE ;
END_PROCESS ;

-- 生産プロセスデータを格納したデータベースに対するインターフェース
INTERFACE manufacturing_schedul_database_interface ;
INHERIT_FROM ( data_resource );
  MESSAGE
    data_change_request( self: PROCESS; user_command : command );
    data_refer_request( self: PROCESS; user_command : command );
    data_change_result( result : LOGICAL );
    request_chield_data( self: PROCESS; user_command : command );
    request_next_chield_process( self : PROCESS; chield : command );
  END_MESSAGE ;
```

```
END_INTERFACE ;

-- 生産データを格納したデータベースに対するインターフェース
INTERFACE manufacturing_database_interface ;
INHERIT_FROM ( data_resource );
  MESSAGE
    data_change_request( self: PROCESS ; user_command : designer_command );
    data_refer_request( self: PROCESS; user_command : designer_command);
  END_MESSAGE ;
END_INTERFACE ;

-- 生産設備／搬送設備に対するインターフェース
INTERFACE machine_tool_interface ; -- interface target machine tool
INHERIT_FROM ( external_environment );
  MESSAGE
    machine_tool.data_transfer(manfacturing_data_id);
  END_MESSAGE ;
END_INTERFACE ;
(*
```

プロセス定義

ここでは，本モデル中で用いられるプロセスを宣言している．具体的には，生産活動における各種のイベントの処理を行う動的生産プロセスモデルである．*)

```
-------------------------------------------------------------------------------
-- Machining/Transporting Process Definition
-------------------------------------------------------------------------------
(*

動的設計プロセスモデルインターフェース定義

*)
-------------------------------------------------------------------------------
-- Machining/Transporting Process Interface
-------------------------------------------------------------------------------
PROCESS ActiveManufacturingProcess
    ( started_from : PROCESS ; info : man_process_info )
INHERIT FROM (clock_interface) ;
MESSAGE

-- 子プロセスの終了
chield_process_ended( chield : PROCESS ; );

-- 機器へのデータ転送の終了
data_transfer_ended( data : manfacturing_data_id )

-- 1秒毎スケジュール確認
BROADCAST every_one_second( current_time : time ):400;

-- 局所変数
LOCAL
    chileds : LIST OF PROCESS;
    self, parent : PROCESS;
    self_info : man_process_info ;
    next_schedule : INTEGER ;
```

```
END_LOCAL;
END_PROCESS ;
(*
```

動的設計プロセスモデルイベント対応定義

```
*)
-------------------------------------------------------------------------------
-- Machining/Transporting Process Event Definition
-------------------------------------------------------------------------------
(*
```

生産活動プロセスの初期化

生産活動プロセスは，初期化時に自身に間する情報を与えられる．この情報に含まれる情報に従って子プロセスの起動又は生産設備に対するデータのダウンロードを行う．

```
*)
-------------------------------------------------------------------------------
-- process initialize
-------------------------------------------------------------------------------
ActiveManufacturingProcess : ActiveManufacturingProcess(
    started_from : PROCESS ;
    info : man_process_info )
BEGIN
    parent := started_from ;
    self_info := info;
    self := SELF_PROCESS(); -- get self process id
    next_schedule := 1 ;
END;
(*
```

一定時間ごとの動作

生産設備は，一定時間毎に自身のスケジュールデータに指定された時間かどうかをチェックし，指定時間に達した場合には，指定された機器へのデータの発信，または子プロセスの起動を行う．

```
*)
-------------------------------------------------------------------------------
-- request to change reference right of data base , called by group manager
-------------------------------------------------------------------------------
ActiveManufacturingProcess :
       BROADCAST every_one_second( current_time : time ):400;
LOCAL  count : INTEGER ;
BEGIN
    IF (current_time == self_info.schedule[ next_schedule ].start_time )
    THEN BEGIN
      IF ( self_info.schedule[ next_schedule ].event_num == CHIELD_START)
      THEN
          for count := 1 to size( chield_process_info )
              ADD( chields , ActiveManufacturingProcess
                             ( self, chield_process_info[count]));
      ELSE IF ( self_info.schedule[ next_schedule ].event_num == DATA_TRANSFER)
          machine_tool.data_transfer(data_id);
    END;

    IF (next_schedule < size(schedule)) THEN
```

```
        next_schedule := next_schedule + 1 ;
END;
(*
```

子プロセスの終了

作業が完了して子プロセスが終了する際のイベント

```
*)
-------------------------------------------------------------------------------
-- chield process is completed and stopped
-------------------------------------------------------------------------------
ActiveManufacturingProcess : chield_process_ended( chield : PROCESS )
BEGIN
    -- 次のスケジュールが無い場合は自身も停止する.
    IF (next_schedule == size(schedule)) THEN BEGIN
         parent.chield_process_ended(self);
         self.halt();
    END;
END;
(*
```

機器へのデータ転送の終了

生産機器へのデータ転送が終了した際のイベント

```
*)
-------------------------------------------------------------------------------
-- chield process is completed and stopped
-------------------------------------------------------------------------------
ActiveManufacturingProcess : data_transfer_ended( data : manfacturing_data_id )
BEGIN
    -- 次のスケジュールが無い場合は自身も停止する.
    IF (next_schedule == size(schedule)) THEN BEGIN
         parent.chield_process_ended(self);
         self.halt();
    END;
END;
(*
```

D　付 録：設計グループ活動モデルと生産活動モデルとの協調活動のEPDLによる記述

付録BとCにおいて記述を行った設計グループ活動モデルと生産活動モデルとの協調活動のモデル化に関する記述を行う.

基本構造

このモデルは，設計作業と製造現場における生産／搬送活動との協調作業を表現するためのプロセスのモデル化を行う.

管理プロセスとして，並行活動の規定などの構造化を定義し，具体的な設計活動／生産活動のプロセスは，その子プロセスとしてBとCで表現した活動プロセスを実行させる.

```
*)
-------------------------------------------------------------------------------
-- Concurrent Design and Manufacturing Process Management Activity Model
-- Rev 1.0, Edited by Yoshihito KIKUCHI
-------------------------------------------------------------------------------
(*
```

型定義

設計設計生産活動管理プロセスのモデル化に対しては，基本的には設計活動／生産活動の両者のプロセスモデルでのデータを用いるが，他の２節と重複する部分に関しての定義は行わない.

```
*)
-------------------------------------------------------------------------------
-- Type Definition
-------------------------------------------------------------------------------
-- プロセスの種類指定のためのフラグ
TYPE process_type : INTEGER ;

-- 設計生産プロセスのスケジュールデータのためのクラス
CLASS design_and_manufacturing_schedule_data ;
    start_time : time ;
    end_time : time ;
    chield_process_type : process_type ;
    chield_process_info : des_and_man_process_info ;
END_CLASS ;

CLASS des_and_man_process_info ;
    SUBCLASS_OF ( design_process_info, man_process_info );
END_CLASS;

(*
```

定数定義

管理プロセスのモデル化では，設計活動／生産活動の両者の管理を目的としているが，具体的な対象に対する入出力は考慮していない，また型定義と同様に他の節と重複する内容の定義は行わない.

```
*)
-------------------------------------------------------------------------------
-- Constant Value Definition
```

```
-------------------------------------------------------------------------------
 DESIGN_PROCESS         : process_type := 1 ;     -- 設計活動プロセス
 MANUFACTURING_PROCESS : process_type := 2 ;     -- 生産活動プロセス
(*
```

インターフェース定義

```
*)
-- 一定時間毎のメッセージを受信するためのインターフェース
PROCESS clock_interface ;
  MESSAGE
    BROADCAST every_one_date( current_time : time ):400;
    BROADCAST every_one_hour( current_time : time ):300;
    BROADCAST every_one_minute( current_time : time ):200;
    BROADCAST every_one_second( current_time : time ):100;
  END_MESSAGE ;
END_PROCESS ;
*)
-------------------------------------------------------------------------------
-- Interface Definition for Design and Manufacturing Management Interface
-------------------------------------------------------------------------------
(*
```

プロセス定義

ここでは，本モデル中で用いられるプロセスを宣言している．具体的には，製造活動における各種のイベントの処理を行う動的生産プロセスモデルである．*)

```
-------------------------------------------------------------------------------
-- Machining/Transporting Process Definition
-------------------------------------------------------------------------------
(*
```

動的設計プロセスモデルインターフェース定義

```
*)
-------------------------------------------------------------------------------
-- Design and Machining Management  Process Interface
-------------------------------------------------------------------------------
PROCESS DesignAndManufacturingManagementProcess
    ( started_from : PROCESS ; info : management_process_info )
MESSAGE

END_MESSAGE;
LOCAL
    chileds : LIST OF PROCESS;
    self, parent : PROCESS;
    self_info : des_and_man_process_info ;
    next_schedule : INTEGER ;
END_LOCAL;
END_PROCESS ;
(*
```

動的設計プロセスモデルイベント対応定義

```
*)
```

```
-------------------------------------------------------------------------------
-- Machining/Transporting Process Event Definition
-------------------------------------------------------------------------------
(*

設計生産活動管理プロセスの初期化

    設計生産活動管理プロセスは，初期化時に自身に間する情報を与えられる．この情報に含まれる
情報に従って子プロセスの起動又は生産設備に対するデータのダウンロードを行う．

*)
-------------------------------------------------------------------------------
-- process initialize
-------------------------------------------------------------------------------
DesignAndManufacturingManagementProcess :
    DesignAndManufacturingManagementProcess(
         started_from : PROCESS ;
         info : management_process_info )
BEGIN
    parent := started_from ;
    self_info := info;
    self := SELF_PROCESS(); -- get self process id
    next_schedule := 1 ;
END;
(*

一定時間ごとの動作

    設計生産管理プロセスは，一定時間毎に自身のスケジュールデータに指定された時間かどうかを
チェックし，指定時間に達した場合には，子プロセスの起動を行う．

*)
-------------------------------------------------------------------------------
-- request to change reference right of data base , called by group manager
-------------------------------------------------------------------------------
DesignAndManufacturingManagementProcess :
       BROADCAST every_one_second( current_time : time ):400;
LOCAL  count : INTEGER ;
BEGIN
    IF (current_time == self_info.schedule[ next_schedule ].start_time )
    THEN BEGIN
      IF ( self_info.schedule[ next_schedule ].event_num == CHIELD_START)
      THEN
          for count := 1 to size( chield_process_info )
              ADD( chields , DesignAndManufacturingManagementProcess
                             ( self, chield_process_info[count]));
      ELSE IF ( self_info.schedule[ next_schedule ].event_num == DATA_TRANSFER)
          machine_tool.data_transfer(data_id);
    END;

    IF (next_schedule < size(schedule)) THEN
         next_schedule := next_schedule + 1 ;
END;
(*
```

子プロセスの終了

作業が完了して子プロセスが終了する際のイベント

```
*)
-------------------------------------------------------------------------------
-- chield process is completed and stopped
-------------------------------------------------------------------------------
DesignAndManufacturingManagementProcess : chield_process_ended( chield : PROCESS )
BEGIN
    -- 次のスケジュールが無い場合は自身も停止する.
    IF (next_schedule == size(schedule)) THEN BEGIN
        parent.chield_process_ended(self);
        self.halt();
    END;
END;
(*
```

参考文献

[Ats93] R Atsumi. A social human activity model as an information infrastructure system. *IFIP Transactions B-14 Application in Technology*, Vol. Proceedings of the JSPE/IFIP TC5/WG5.3 Workshop on the Design of Information Infrastructure Systems for Manufacturing, DIISM '93, pp. 171–182, 1993.

[BA91] D. BEN-ARIEH. Concurrent modeling and simulation of multi-robot systems. *Robotics & Computer-Integrated Manufacturing*, Vol. 8, No. 1, 1991.

[BA92] D. Ben-Aroeh. Concurrent modeling language (cml) for discrete process modeling, simulation and control. *Journal of Intelligent Manufacturing*, Vol. 3, pp. 31–41, 1992.

[Bai] Ian Bailey. *ISO TC184/SC4/WG5/N51 EXPRESS-M : Reference Manual.* ISO/TC 184/SC 4.

[BB93] Frank Biemans and Pieter Blonk. On the formal specification and verification of cim architectures using lotos. *Computers in Industry 7*, pp. 491–504, 1993.

[BK85] J.A. Bergstra and J.W. Klop. *Algebra of Communicating Process with Abstruction*, pp. 277 121. North-Holland, 1985.

[Bud92] Timothy A. Budd. オブジェクト指向言語プログラミング入門. トッパン, 1992.

[Car90] Steeb Carlson. VHDLによるHDL設計と論理合成入門. エス・シー・ハイテクセンター株式会社, 1990.

[CCI88] CCITT. *CCITT Recommendation Z.100: Specification and Description Language SDL (Blue Book, Volume X.1 - X.5).* CCITT General Secretariat, 1988.

[Cla93] D.P. Clausing. World-class concurrent engineering. *NATO ASI Series F: Computer and Systems Sciences*, p. 1, 1993.

[Cle92] K.J Cleetus. Definition of cucurrent engineering. *CERC Technical Report Series*, Vol. CERC-TR-RN-92-003, , 1992.

[Cle93] K.J. Cleetus. Virtual team framework and support technology. *NATO ASI Series F: Computer and Systems Sciences*, p. 41, 1993.

[Davi93] David A. Taylor, 増永良文監訳, 寺島哲史訳. Higher Education Computer Series 2 オブジェクト指向アプローチ. アジソン・ウェスレイ, 1993.

[DH92] R. Duschl and N.C. Hopkins. *SSADM & GRAPES - Two Complementary Major European Methodologies for Information Systems Engineering.* Springer-Verlag, Berlin, 1992.

[Esp93] Esprit Consortium AMICS Eds. *Reserch Reports ESPRIT Project 688/5288 AMICE Volume 1, CIMOSA: Open System Architecture for CIM.* Springer-Verlag, 1993.

[FM94] Winfried Felser and Wolfgang Muller. Express-p: Extending express for process modeling and monitoring. *Engineering Data Management: Integrating the Engineering Enterprise*, pp. 145–153, 1994.

[Fow95] Julian Fowler. *STEP for Data Management Exchange and Sharing.* Technology Appraisals, 1995.

[GR89] N. Gehani and W.D. Roome. *The Concurrent C Programming Language.* AT&T Bell Telephone Laboratories, Inc., 1989.

[HM93] Satoshi Hamada and Takizawa Makoto. Transaction model of automated guided vehicles. *IFIP Transactions B-14 Application in Technology*, Vol. Proceedings of the JSPE/IFIP TC5/WG5.3 Workshop on the Design of Information Infrastructure Systems for Manufacturing, DIISM '93, pp. 413–424, 1993.

[Hoa85] C.A.R. Hoare. *Coomunicating Sequential Process: Prentice-Hall International Series in Computer Science.* Prentice-Hall UK Ltd., 1985.

[Hoa92] C.A.R. Hoare. ホーアＣＳＰモデルの理論. 丸善株式会社, 1992.

[IBM] IBM Canada. *OSI/Manufacturing Message services Programmer's Reference SC09-1305.* IBM Canada.

[ISO89] ISO. *Information Processing systems - Open system Information - LOTOS - A Formal Description Technique Based on the Tmporal Ordering of Observational Behavior.* ISO, 1989.

[ISO91] ISO/TC97/SC21. *IPS-OSI-ESTELLE - A Formal Description Technique based onan extended state transition model - ISO 9074.* ISO, 1991.

[ISO95a] ISO/TC 184/SC 4. *Industrial automation systems - Product data representation and exchage Part 1 : Overview and fundamental principles.* ISO/TC 184/SC 4, 1995.

[ISO95b] ISO/TC 184/SC 4. *Industrial automation systems - Product data representation and exchage Part 11 : Description Methods : The EXPRESS Language Reference Manual.* ISO/TC 184/SC 4, 1995.

[ISO95c] ISO/TC 184/SC 4. *Industrial automation systems - Product data representation and exchage Part 21 : Implementation methods : Clear Text Encoding of the Exchange Structure.* ISO/TC 184/SC 4, 1995.

[ISO95d] ISO/TC 184/SC 4. *Industrial automation systems - Product data representation and exchage Part 201 : Application Protocol : Explicit Draughting Volume 1, Annex-D Application Activity Model.* ISO/TC 184/SC 4, 1995.

[ISO95e] ISO/TC 184/SC 4. *Industrial automation systems - Product data representation and exchage Part 41 : Integrated generic resources : Fundamentals of product description and support.* ISO/TC 184/SC 4, 1995.

[ISO95f] ISO/TC 184/SC 4. *Industrial automation systems - Product data representation and exchage Part 42 : Integrated generic resources : Geometric and topological representation.* ISO/TC 184/SC 4, 1995.

[ISO95g] ISO/TC 184/SC 4. *Industrial automation systems - Product data representation and exchage Part 43 : Integrated generic resources : Representation structure.* ISO/TC 184/SC 4, 1995.

[ISO95h] ISO/TC 184/SC 4. *Industrial automation systems - Product data representation and exchage Part 44 : Integrated generic resources : Product structure configuration.* ISO/TC 184/SC 4, 1995.

[ISO95i] ISO/TC 184/SC 4. *Industrial automation systems - Product data representation and exchage Part 46 : Integrated generic resources : Visual presentation.* ISO/TC 184/SC 4, 1995.

[ISO95j] ISO/TC 184/SC 4. *Industrial automation systems - Product data representation and exchage Part 22 : Implementation methods : Stndard Data Acces Interface.* ISO/TC 184/SC 4, 1995.

[Jame89] James R. Armstrong, 白石繁, 山本 靖訳. VHDLデザインテクニック標準ハードウェア記述言語とチップレベル・モデリング. 電波新聞社, 1989.

[Kim93] Fumihiko Kimura. A computer-supported framework for concurrent engineering - an approach based on virtual manufacturing. *IFIP Transactions B-14 Application in Technology*, Vol. Proceedings of the JSPE/IFIP TC5/WG5.3 Workshop on the Design of Information Infrastructure Systems for Manufacturing, DIISM '93, pp. 345–359, 1993.

[KR98] Brian W. Kernighan and Dennis M. Ritchie. *The C Programming Language, second edition.* Printice-hall, 1998.

[LSU89] R. Lipsett, C. Shaefer, and C Ussert. VHDL 言語記述によるハードウェア設計へのアプローチ. マグロウヒル出版株式会社, 1989.

[MAE+95] A. Molina, A.H.AI-Ashaab, T.I.A. Ellis, R.I.M Young, and R. Bell. A review of computer-aided simultaeous engineering systems. *Reserch in Engineering Design*, Vol. 7, No. 1, pp. 38–63, 1995.

[MFF92] Wolfgang Muller, Winfried Felser, and Buijs Frank. Express meets process modeling. *Proceeding of EUG'94 (Express Users Group '94)*, 1992.

[Mil89] Robin Milner. *Communication and Concurrency: Prentice-Hall International Series in Computer Science.* Prentice-Hall UK Ltd., 1989.

[MMPd92] Richard J. Mayer, Capt. Michael, K. Painter, and Paula S. deWitte. *IDEF Family of Methods for Cuncurrent engineering and Business Re-engineering Applications.* Knowledge Based Systems,Inc., 1992.

[MSJ] K. Mertins, W. Suggenguth, and R. Joehen. Planning of enterprise related cim structures. *Pascalstrasse 8-9.*

[NISa] NIST. *Processing Standards Publication 183, Integration Definition for Functional Modeling (IDEF0).* NIST.

[NISb] NIST. *Processing Standards Publication 184, Integration Definition for Functional Modeling (IDEF1X).* NIST.

[PK89] Dhiraj K. Pathak and Bruce H. Krogh. Concurrent operation specification language, cosl, for low-level manufacturing control. *Computers in Industry*, pp. 107–122, 1989.

[PST95] B. Potter, Jane Sinclair, and David Till. ソフトウェア仕様記述の先進技法－Z言語. 株式会社トッパン, 1995.

[RHC90] Kent Reed, Dennette Jr. Harrod, and William Conroy. *The Initial Graphics Exchange Specification (IGES) Version 5.0.* U.S. Department of Commerce, National Institute of Standards and Technology, September 1990.

[RZP89] M Roboam, M Zanettin, and L Pun. Grai-idef0-merise (gim): Integrated methodology to analyze anddefign manufacturing systems. *computer-integrated Manufacturing Systems*, Vol. 2, No. 2, pp. 114–127, 1989.

[Tho94] V. Thomson. Process management applied to manufacturing. *Proc. 10th ISPE/IFAC Int. Conf. on CAD/CAM Robotics and Factories of the Future*, 1994.

[U.S83] U.S. Air Forces. Integrated computer-aided manufacturing architecture icam project part ii, vol. iv - function modeling manual (idef0). *computer-integrated Manufacturing Systems*, 1983.

[Ver93] F.B. Vernadat. Cimosa: Enterprise modeling and enterprise integrtion using a process based approach. *Proc. JSPE/IFIP WG5.3 Workshop on The Design Information Infrastructure Systems for Manufacturing (DIISM'93)*, pp. 1–15, 1993.

[Ver94] F.B. Vernadat. A language for cim system specification. *Proc. 10th ISPE/IFAC Int. Conf. on CAD/CAM Robotics and Factories of the Future*, pp. 283–288, 1994.

[W.88] W. ライシッヒ. ペトリネット理論入門. シュプリンガー・フェアラーク東京株式会社, 1988.

[Weg92] P. Wegner. はやわかりオブジェクト指向. 共立出版株式会社, 1992.

[Wen] Ji Wenn. *ISO TC184/SC4/WG5/N251(P2) EXPRESS-V : Reference Manual.* ISO/TC 184/SC 4/WG 5.

[WPBS88] R.I Winner, J.P. Pennel, H.E. Bertrand, and M.M.G. Slusarczuk. The role of concurrent engineeringin weapon system acquisition. *IDA-Report R-338*, 1988.

[ナイ89] ナイレンゲハーニ, ウィリアムルーム. Concuurent C プログラミング言語 : 平行プログラミング入門. 株式会社トッパン, 1989.

[伊藤 92] 伊藤公俊他. 抽象実体記述言語による transformable prototype の構成. 第 10 回設計シンポジウム講演論文集, 1992.

[奥川 95] 奥川峻史. ペトリネットの基礎. 共立出版株式会社, 1995.

[岡田 90] 岡田康治. Estelle 言語の特質と処理系の現状と動向. 情報処理, Vol. 31, No. 1, 1990.

[岡田 92] 岡田公治他. 設計意図モデルと設計過程記述言語に基づく計算機援用設計. 第 10 回設計シンポジウム講演論文集, 1992.

[岩田 95] 岩田一明, 荒井栄司, 小野里雅彦, 藪崎清・ヒベルト. 設計・生産におけるコンカレントエンジニアリングのためのコミュニケーションシステムに関する研究 (第一報, 複数ユーザ間での会話モデルに基づくコミュニケーションシステムの実現). 日本機械学会論文集 (C 編), pp. 392–397, 1995.

[菊地 92] 菊地慶仁, 岸浪建史, 斎藤勝政. 3 階層スキーマ構造に基づくＣＡＤデータ交換. 精密工学会誌, Vol. 58, No. 1, 1992.

[菊地 93] 菊地慶仁, 田中文基. 形式的データ仕様記述言語 express. 精密工学会誌, Vol. 59, No. 12, pp. 19–24, 1993.

[菊地 95a] 菊地慶仁, 岸浪建史. 設計生産情報モデルの記述と表現に関する研究第 1 報　基本概念と形式記述言語. 精密工学会誌, Vol. 61, No. 5, 1995.

[菊地 95b] 菊地慶仁, 岸浪建史. 設計生産情報モデルの記述と表現に関する研究 第 3 報並行活動シュミレータの開発. 精密工学会誌 投稿中, 1995.

[菊地 96] 菊地慶仁, 岸浪建史. 設計生産情報モデルの記述と表現に関する研究 第 2 報設計生産活動の統合モデル. 精密工学会誌 投稿中, Vol. 62, No. 3, 1996.

[江沢 95a] 江沢智. Pdm 入門, 第 1 回, 複雑な製品情報を管理する pdm システムの基本性能. 日経コンピュータグラフィックス, pp. 200–208, 7 1995.

[江沢 95b] 江沢智. Pdm 入門, 第 2 回, pdm システムのアーキテクチャ. 日経コンピュータグラフィックス, pp. 231–236, 9 1995.

[江沢 95c] 江沢智. Pdm 入門, 第 3 回, 業界標準や規格をフルに使う現実的な pdm 構築. 日経コンピュータグラフィックス, pp. 199–206, 10 1995.

[高橋 90a] 高橋薫, 神長裕明, 白鳥則郎. Lotos 言語の特質と処理系の現状と動向. 情報処理, Vol. 31, No. 1, 1990.

[高橋 90b] 高橋浩, 中川路哲男. 構文定義用言語 asn.1 の特質と処理系の現状と動向. 情報処理, Vol. 31, No. 1, 1990.

[高橋 93] 高橋薫, 大蒔和仁. プロセス代数とその応用 (9) lotos - 実用思考のプロセス代数基礎編. *bit*, Vol. 24, No. 3, 1993.

[今村 92] 今村聡. 機械設計用制約オブジェクト指向言語 fdl. 第 10 回設計シンポジウム講演論文集, p. 1, 1992.

[藪崎 93b] 藪崎清・ヒベルト他. 設計・生産におけるコンカレントエンジニアリングのためのコミュニケーションシステムに関する研究（第３報）－コミュニケーションにおける対象表現－. 1993 年度精密工学会秋季大会学術講演会論文集, pp. 15–16, 1993.

[藪崎 93c] 藪崎清・ヒベルト他. 設計・生産におけるコンカレントエンジニアリングのためのコミュニケーションシステムに関する研究（第４報）. 1993 年度精密工学会秋季大会学術講演会論文集, pp. 17–18, 1993.

[藪崎 95] 藪崎清・ヒベルト, 岩田一明, 荒井栄司, 小野里雅彦. 設計・生産におけるコンカレントエンジニアリングのためのコミュニケーションシステムに関する研究 (人間中心コンカレントエンジニアリングのための対象表現). 日本機械学会論文集 (C 編), pp. 470–475, 1995.

[鈴木 93] 鈴木宏正. Step における製品モデルの構造. 精密工学会誌, Vol. 59, No. 12, pp. 11–18, 1993.

[今村 94] 今村聡. 工業設計支援用対象指向言語の開発. 精密工学会誌, Vol. 60, No. 9, pp. 1242–1246, 1994.

[若原 90] 若原恭. Sdl 言語の特質と処理系の現状と動向. 情報処理, Vol. 31, No. 1, 1990.

[勝山 90] 勝山光太郎, 佐藤文明. 試験仕様記述言語 ttcn の特質と処理系の現状と動向. 情報処理, Vol. 31, No. 1, 1990.

[小島 93] 小島俊雄, 中村伊知郎, 久貝穣, 木村文彦. Step を用いた cad データ交換システムの定式化. 精密工学会誌, Vol. 59, No. 2, pp. 61–66, 1993.

[森野 94] 森野和好, 戸部美春. コンピュータ通信シリーズ 2 プロトコル構文規定言語-ASN.1. カットシステム, 1994.

[水野 90] 水野忠則. 標準仕様記述言語の概観. 情報処理, Vol. 31, No. 1, pp. 20–22, 1990.

[水野 94] 水野忠則監修. プロトコル言語. カットシステム, 1994.

[村山 94] 村山洋司. 欧米における pdm システム活用事例. 精密工学会, 第 209 回講習会講演論文集, テキスト, 設計リエンジニアリングの鍵を握る PDM（製品データ管理）システムの実用事例, pp. 5–10, 1994.

[仲 義 94] 仲 義輝他. Concurrent engineering における意志決定構造の分析. 1994 年度精密工学会秋季大会学術講演会論文集, pp. 969–970, 1994.

[椎塚 92] 椎塚久雄. 実例ペトリネット-その基礎からコンピュータツールまで-. コロナ社, 1992.

[渡辺 95] 渡辺一正. 特集 cals が実現するデータ変換のない企業連携. 日経コンピュータグラフィックス, pp. 94–103, 5 1995.

[二木 90] 二木厚吉. Iso における形式記述技法の標準化動向. 情報処理, Vol. 31, No. 1, 1990.

[二木 92] 二木厚吉, 富樫敦. プロセス代数とその応用 (1) 形式仕様とプロセス代数. *bit*, Vol. 23, No. 11, 1992.

[富樫 93a] 富樫敦, 木村成伴. プロセス代数とその応用 (5) ccs. *bit*, Vol. 24, No. 2, 1993.

[富樫 93b] 富樫敦, 木村成伴. プロセス代数とその応用 (6) csp. *bit*, Vol. 24, No. 3, 1993.

[平田 92] 平田昌信. 設計環境に革命をもたらす pdm. 日経コンピュータグラフィックス, pp. 70–85, 10 1992.

[木村 94] 木村文彦. ＰＤＭシステムとコンカレントエンジニアリング. 精密工学会, 第 209 回講習会講演論文集, テキスト, 設計リエンジニアリングの鍵を握る PDM（製品データ管理）システムの実用事例, pp. 1–4, 1994.

[木村 95] 木村文彦・小島俊雄. 製品モデル表現とその利用技術 STEP. 日本規格協会, 1995.

[薮崎 93a] 薮崎清・ヒベルト他. 設計・生産におけるコンカレントエンジニアリングのためのコミュニケーションシステムに関する研究（第２報）－複数ユーザ間でのコミュニケーションの実現－. 1993 年度精密工学会春季大会学術講演会論文集, pp. 687–688, 1993.

inches 1 2 3 4 5 6 7 8

cm 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19

Kodak Color Control Patches

© Kodak, 2007 TM: Kodak

Blue Cyan Green Yellow Red Magenta White 3/Color Black

Kodak Gray Scale

C Y M

© Kodak, 2007 TM: Kodak

A 1 2 3 4 5 6 **M** 8 9 10 11 12 13 14 15 **B** 17 18 19