JSD 概論

加藤潤三
（株）コムニック創研

JSD(Jackson System Development)の概要をモデル化と機能の仕様化を中心に説明する．JSD の特徴はソフトウェアシステムの開発を仕様化とその実現に明確に分離していることである．JSD の成果物という側面から見たときの特徴としてこの分離に適切な仕様書の構成と構造がある．JSD のモデル化で採用しているアプローチは系統的であり,現在でもそのアプローチは有用である．今まであまり述べられなかった仕様書の構成と構造の特徴とモデル化それから JSD の限界と現時点での応用について述べる．

JSD Overview

Junzo KATO
COMNIC-SOKEN CORP.

This paper talks about an overview of JSD(Jackson System Development) and focuses on JSD's modeling and network phase. The feature of JSD is having divided development of a software system into specification and its implementation clearly. The composition and structure of the specifications of JSD is suitable for this separation. The approach adopted by modeling of JSD is systematic, and the present of the approach is also useful. This paper discussed the composition of specifications and the feature of structure which were seldom described until now, modeling, the limit and an application of JSD .are discussed.

1. はじめに

JSP（Jackson Structured Programming）[1]と JSD（Jackson System Development）[2],[3],[4]をまとめてジャクソン法と言うことが多い．JSP は JSD の一部になっていることと，JSD は逐次プロセスからなるシステムを如何に組み立てるかという点で開発者をガイドし，JSP はその逐次プロセスの設計で使われるということに由来する. JSP はその応用範囲が列構造を持つデータ処理に限られデータ構造が与えられないと設計できない．JSD は JSP の背景にある考えを継承し応用範囲を拡張したものである. JSD の実現フェーズでは JSP のいくつかの手法(Technique)を使う. JSP と JSD はいずれもモデル化，機能の仕様化，仕様の実現というフェーズからなるアプローチを採用している．

ジャクソンは技法(Method)は次の特性を持つべきであると述べている

(1) 開発の手順とその成果物が明確であり，手順の終了基準が明示されている．
(2) 各々の作業を無駄なく円滑に進めるための適切な道具が用意されている．
(3) 開発途中の決定とその順序を開発方法は持つ．決定は陽に記述され，次の段階で利用できるようになっている．

JSD と JSP はこれらの特徴を持っている．JSD を理解するにはまず次に述べるその原理を理解することが大切である．

(1) 別個に順序付けられたタスクに開発作業を分割し，各々のタスクに対して適切なツールと完了基準を与えるのが開発方法である．
(2) システムが行う機能よりもシステムが依存する実世界のモデル化の考慮から始める．

(3) 動的な実世界は動的なモデルで記述する．静的な実世界は静的なモデルで記述する．
(4) 設計は（仕様化）は実現に先行する．
(5) 実現は正当性を保存する機械的な変換によることが理想である．

JSD は 1980 年代に開発されたソフトウェア開発技法である. Client/Sever アーキテクチャに代表される分散システムや WWW のようなインターネット上のシステムについては言及しようがない．しかし JSD の仕様書のアーキテクチャは Client/Sever アーキテクチャの 3 層モデルにおけるリソース層と機能層に対応可能な構造を持っている．この点については後で説明する．JSD が開発された年代は構造化分析/設計がソフトウェア開発技法の主流であった. オブジェクト指向に関してはプログラミング言語および開発/実行環境である SmallTalk80[12]が話題になり始めたばかりである．JSD が与えたソフトウェア開発技法への影響は大きく，1980 年代後半の構造化分析/設計[13]，1990 年代のオブジェクト指向分析/設計[14]に影響を与えた. 最近でもオブジェクト指向分析/設計で応用されている[15]などその息は長い．特に，実世界の実体(Entity)を抽出する感度が良く,モデル化のアプローチの系統的なところが今でも注目されている．

JSD が開発された背景とその本質を理解するには JSP を JSD の説明の導入で使うのが良いと言われている．紙面の都合から JSP の説明を省き JSD の特徴的な所を中心に JSD そのものの概要を説明する. 説明する順序と内容を次に述べる．

(1) 2．JSD の紹介：JSD 仕様書のアーキテクチャ，JSD のモデル，JSD 仕様書の２つのビュー，JSD の開発フェーズ．
(2) 3．モデル化：実世界の抽象化，抽象化した実世界の具体化（モデル化）．
(3) 4．機能の仕様化：並列逐次プロセスのネットワーク，プロセス間の通信，仕様化する機能の分類．
(4) 5．仕様の実現：JSD における実現の考え方の簡単な紹介．
(5) 6．適用範囲と応用：JSD の限界と例を使った応用についての検討．
(6) 7．まとめ：JSD の課題と応用にあたっての注意点

2. JSD の紹介

JSD を開発したジャクソンやキャメロンは実世界のモデルとしていきなり実体関連図[17]を作成することには無理があるという考えを持っていたと思われる．1970 年の後期から 1980 年代の初期にかけて, 計算機システムの開発において実世界のモデルが重要な役割をはたすこととその必要性が知られ始めた頃であったので, 実世界のシミュレーションモデルとしての JSD のモデルは新鮮であった．JSD のモデルは計算機で持つ実世界の類似したモデル(Analogic Model)である．計算機上の実世界の類似したモデルがその解釈に計算モデルを使うのは自然である．JSD のモデルは実世界の動的側面を並列逐次プロセス（並列の動作する逐次プロセス）で記述したものであり,背景には CSP[11]で提案された並列処理の概念と計算モデルがある．JSD と CSP とはほぼ同時期に開発されており JSD におけるプロセス間通信の意味は CSP そのものではないが CSP で表現できる．実世界は並存する実体の集まりからなりそれらが互いに通信しているという見方が JSD が捉える実世界である．

JSD には次のような主張がある．

(1) 実世界の主題領域（Subject of Domain）をモデル化する．実世界のモデルは機能要求よりも安定している．
(2) 実世界のモデルの記述は静的側面の記述では不足である．動的な実世界には動的側面の記述が良い．
(3) 動的側面は，実体と実体の振る舞いを出来事の列構造で記述する．実体は併存しこれらが並列動作し互いに通信するのであるから，モデルとして表現するには並列逐次プロセスによるモデルが適している．この並列逐次プロセスの制御構造をジャクソン木で表現する．
(4) モデルによって利用者はシステムで何ができるのかを知ることができる．
(5) モデルを使い機能を定義しシステムで維持すべきデータを仕様化する．
(6) 仕様化の段階でシステムの構成要素を明示的に書き尽くす．
(7) 実現段階でシステムの稼働環境に合うように再構成する．実現は仕様書の再構成と変換からなる．

(8) 隠された決定をなくすような仕様書の構成と開発手順，実現手法である．実世界のモデル化により課題の範囲は明示的に決まり，モデルを使った機能の仕様化によって機能は明示的に記述され，仕様書の変換による実現によって実現時の隠された決定をなくす．

JSD のモデルは動的側面をモデル化したもので，静的側面はそれから導出できる．JSD は動的な側面を持つ世界を対象としているからである．時間軸に沿って変化する実体が JSD のモデル化の対象であり，これをまず抽出する．実世界の実体を具体化したものがモデルであり，JSD では並列逐次プロセスであるモデルプロセスとして記述する．実体の属性は各々の並列逐次プロセスの状態の一部になる．

ジャクソン木を並列逐次プロセスの構造として使うのは，出来事の列に関心があり，途中の状態には関心はないからである．なお，手戻り（Backtracking）については実体の構造では選択または繰り返し構造で示し，並列逐次プロセスにおいて始めて明示的に示す．これは JSP の認識困難問題へのアプローチと同じである．JSD では手戻りの表記法が用意されている．図１でジャクソン木を示す．

JSD の特徴を現代風にとらえなおすと，仕様書のアーキテクチャ，実体の抽出アプローチ，実体の抽出からモデル化と機能の仕様化の系統的アプローチを挙げることができる．JSD モデルの計算モデルが存在し原理的に仕様が実行可能であることが一因となっている．本章では仕様書のアーキテクチャを説明し，実体の抽出アプローチ，実体の抽出からモデル化の系統的アプローチを３章のモデル化で述べ，機能の仕様化の系統的アプローチを４章の機能の仕様化で説明する．

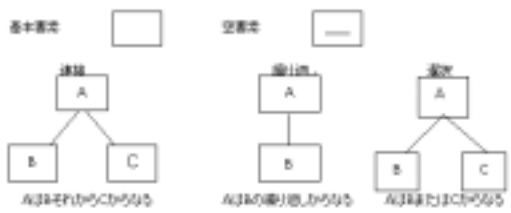

図１ ジャクソン木（Jackson Tree）

2.1 JSD の仕様書のアーキテクチャ

JSD の仕様書は図 4 で示すように並列逐次プロセスのネットワークと個別のプロセスの仕様書の２つのレベルで記述される．並列逐次プロセスのネットワークは並列逐次プロセスを無限長の待ち行列（JSD ではデータストリーム結合）と並列逐次プロセスの状態の要求（JSD では状態ベクトル結合）の２種類の結合方式で結び付ける．このネットワーク図の表記法を図 2 で示す．

図２ JSD ネットワーク図の表記法

JSD の仕様書は入力サブシステム，モデル，機能の３部から構成されている．それらの繋がりを図３で示す．図 5 で示すように JSD の仕様書はモデルプロセスを核として，回りに機能を仕様化した機能プロセス（情報機能、会話機能），モデルプロセスまたは機能プロセスの外界とのインタフェースを仕様化した入力サブシステムの入力プロセスからなる．

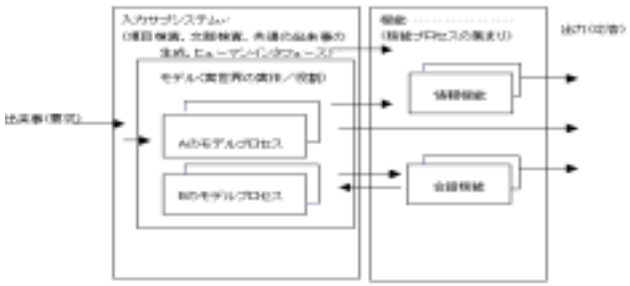

図 3 JSD の仕様書の構成

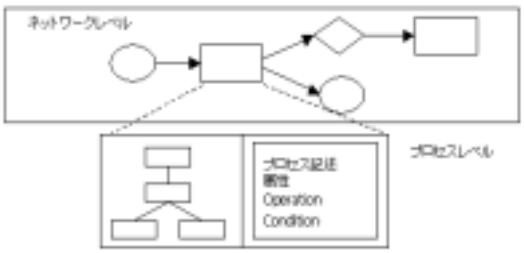

図 4 JSD の２レベル仕様書

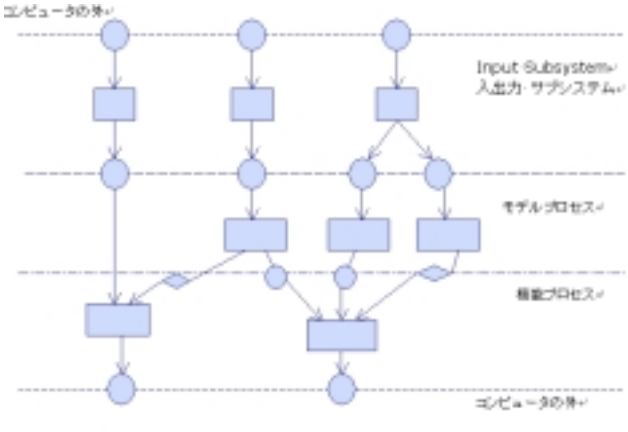

図5 JSDにおけるプロセスネットワークの構造

2.2 JSD の手順

JSD はソフトウェアシステムの仕様化と仕様の実現からなる. 仕様化はさらにモデル化フェーズとネットワーク作成フェースに分かれる. ここでは仕様化を中心に説明し実現フェーズについては簡単に触れることにする.

（1） モデル化フェーズ

実世界を抽象化しそれをモデルとして記述（具体化）する. ここで記述されたモデルは機能の仕様化の終了まで維持されつづける. システムアーキテクチャをこのフェーズで反映する.

（1）－1　実世界の記述
　　動作とその属性の記述
　　実体の構造

（1）－2　実体のプロセス化
　　実体の属性の記述
　　実体の構造の詳細化

（2） ネットワーク作成フェーズ

機能の仕様化のフェーズであり, 機能別にネットワークを作成する. モデルプロセスの構造は崩さないのが原則である.

以下の種類のプロセス（機能プロセス）を追加する. 追加は, ネットワークの変更・追加, プロセス内部構造の変更・追加, 時間上の制約（タイミング制約）の記述がある. ここでいう変更とは, すでに記述されているプロセスの変更を言う. JSP を応用しプロセスを追加する.

1） 情報機能プロセス
2） 会話機能プロセス
3） 入力プロセス

（3） 実現フェーズ

仕様書を再構成し変換する.

（3）－1　実際の実行環境に適した物理設計
　　プログラム設計
　　物理データ設計（データベース設計も含む）

（3）－2　プログラム作成とテスト

3. モデル化（モデル化フェーズ）

JSD のモデル化とは, 実世界の離散系シュミレーションモデルを作ることを言う. システムのスコープはモデルによって定まる. データモデルの作成にも使える JSD のモデルはデータ項目の更新手続きとその時間順序に関する制約, およびデータの整合性の制約を 記述するように拡張したデータモデルの一つである.

JSD ではモデルプロセスを実体とその役割で識別する. ここで言う役割とは実体の意味ある振舞い, 即ち意味を持つ実体が受け入れる出来事の列, で実体を分離する. 以下, JSD の実体を実体/役割として述べることにする.

実体／役割の構造の解像度は, その振舞いを定義する出来事の精密さに依存する. この精密さは機能要求の背景にあるビューに関連する.

機能を仕様化するネットワーク作成フェーズに入る前に, システムのアーキテクチャの決定を反映し, アーキテクチャに沿ってモデルプロセスを分割する.

3.1 実世界の記述

(1) 動作(action)に着目する.

JSD では, 実体/役割を抽出する少し前に, 関心のある動作を抽出する. 動作の候補から課題領域にあると想定される動作を選択しそれらの動作の説明とその属性を記述する。属性のデータ型を与える. 動作をモデルプロセスへの出来事（Event）として扱う.

図 6 動作一覧

(2) 実体の候補を選択する.

動作の説明とその属性から実体の候補を抽出する. この時点では役割については明確ではない. 課題領域内にありかつ動的側面を持つ実体を選択する.

図 6 の例の動作の定義の文から動作の主体または動作の対象を抽出する．これが実体の候補になる．図 6 からは図書，図書館，利用者が実体候補である.

(3) 実体の構造（振舞い）を定義する.

選択された実体の振舞いを動作の列としてジャクソン木で表現する．表現した結果が JSP における入り交じり不一致のように構造として意味を持たない場合, 複数の構造が重なっているかどうかを調べ，重なっているときは分離する. 分離した構造は実世界における役割を持っているので役割名を与え実体/役割として定義する.

(4) 共通動作をまとめる.

複数の実体/役割で共有される動作は共通動作としてまとめる. 共通動作は複数のモデルプロセスを同期させる出来事を示す. ここでは共通動作を共通の出来事としてまとめている.

3.2 実体のプロセス化

(1) 実体の属性の記述

実体/役割の属性を識別子，それから受け入れる動作の属性から決める. 少なくとも，モデルプロセスの識別子と共通動作がある場合は別のモデルプロセスの識別子を決める. 属性にはそのデータ型も同時に与える.

(2) 属性を与えた実体/役割をモデルプロセスとして定義する.

操作, 条件を追加し実体の構造の詳細化しモデルプロセスとする. 動作はその属性のデータ型を定義した後, モデルプロセスへの出来事として定義する．図 9,図 10 で例を示す.

3.3 データ型の定義

動作と実体/役割の属性の定義で宣言したデータ型で基本的なデータ型（整数，文字列，浮動小数点，文字など）以外の仕様書で新たに定義するデータ型を定義する．抽象データ型でも良い.

3.4 システムアーキテクチャの反映

ハードウェア，ネットワーク，ソフトウェアのアーキテクチャが決まっていればそれを反映する. 反映した結果は初期モデルプロセスの分散やサブシステムへの分割となる.

3.5 成果物：

初期モデル:機能の仕様化でモデルプロセスを完成させるので初期プロセスという.

JSD のプロセス定義

共通の出来事（共通動作）一覧表

データ型の定義

JSD の手順と成果物には，仕様書の作成途中で発生する手戻りを最小にするために同じ表記法の採用と不完全なものを徐々に組み立て上げるという工夫がある. 最終的に組み立てるものが見えているため, 最初の分析は組み立てに必要最小限なものにしている.

図 7 JSD モデルの位置付け

共通の出来事（図書館システム）

出来事	実体	説明	属性	型
貸出	利用者 図書	利用者の求める図書が貸出可能な場合, 利用者にそれを出す.	図書識別子 貸出日付 利用者	図書ID 日付 利用者ID
返却	利用者 図書	利用者は借りていた図書を返却する.	図書識別子 返却日付 利用者	図書ID 日付 利用者ID

図 8 共通の出来事（共通動作）一覧表

図 9 モデルプロセス定義（構造図）

図 10 モデルプロセス定義（その他）

4. 機能の仕様化（ネットワーク作成フェーズ）

モデルプロセスを元に, 個別の機能を仕様書に追加する. ここで言う機能とは出力を作るためにシステムが実行する動作を言う. モデルプロセスは機能の追加によって属性とか操作が追加される. JSD の後期[3]にいくつかのプロセス間通信のプリミティブが追加されたが, 基本的にはプロセス間通信はデータストリーム結合と状態ベクトル結合の２種類である. データストリーム結合がマージされているときは複数の並列逐次プロセス間通信なので複数のプロセスからの通信文の到着には非決定性が発生する. 非決定性をそのまま表現したラフマージによる結合がある. 非決定性を避けるには固定マージとして個々のデータストリームを個別に読み込む. 機能プロセスの追加で JSP を応用する.

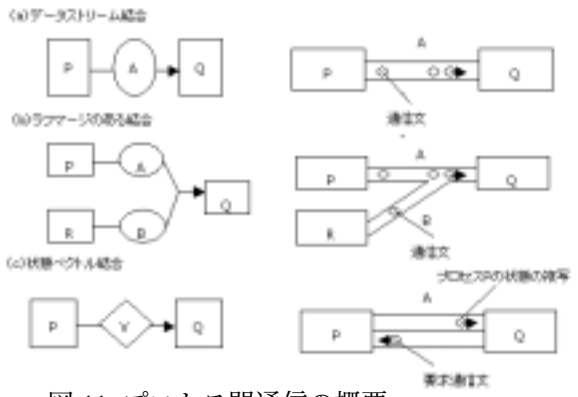

図 11 プロセス間通信の概要

機能の仕様化によって, 離散系シュミレーションモデルに報告書機能とモデルへのフィードバック機能, 入力サブシステム（入力検査, ユーザーインターフェース, 整合性の維持）を追加し, 仕様書を完成させる. 完成した仕様書は原理的には実行可能である. 最終的には JSP 基本方法で解決できるプロセスまで機能を展開する.

4.1 機能の種類

出来事を契機に出力を作成する機能を情報機能という. 情報機能については, 出力を作成する操作をモデルプロセスに埋め込むか出力を作成する情報機能プロセスを追加する. 情報機能の他に, モデルへの出来事を生成する対話機能プロセスを追加する対話機能, 入力データをモデルプロセスまたは機能プロセスへ出来事として渡す入力サブシステムがある. 入力プロセスからなる入力サブシステムは外部からの物理的な入力データを変換する他にデータの誤りを検出しモデルに誤まった出来事を渡さないようにする役割も持つ.

情報機能プロセス, 対話機能プロセス, 入力プロセスのいずれもその生成から消滅まで個別に属性を持つ. このプロセスと属性が図 7JSD モデルの位置付けの機能から導出されたモデルに反映される.

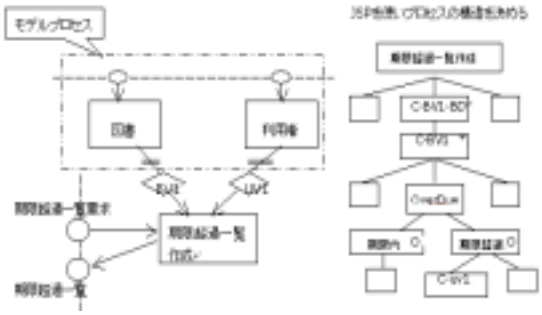

図 12 モデルプロセスと機能プロセス

4.2 成果物：

ネットワーク図

（ＳＳＤ：System Specification Diagram）

機能要求別（タイミング付）

全体（またはグループ別）

JSD プロセス定義

識別子, 説明文, 属性とその型, 入力/出力データストリーム, 出来事とその説明, 操作, 条件, 状態ベクトル参照操作と構造図,

データ型定義
　データ型の定義
　抽象データ型も定義できる.
共通の出来事（共通動作）一覧表
　モデル化で作成したもの.
その他
　ネットワークフェーズの終了時には, ユーザインターフェース・出力レイアウト（ドラフト）が決定していることを想定している.

図 13 情報機能プロセスの例

5. 仕様の実現(実現フェーズ)

JSD は仕様の実現についての完全な手法を提供していない. 実現はその道の専門家に任せることを前提にしている．JSD のこの性質はその応用範囲を広くし現在でも応用可能な技法としている一つの要因である.

JSD における実現とは，実行可能な仕様書を実際の実行環境に合わせて効率良いシステムに再構成し変換することである. 実装環境のプラットフォームによって実現方法が異なるので, 仕様の変換手法のみを提示している. 変換手法には, JSP から継承した手法（プログラム転換, 状態ベクトル分離，プログラム分割）の他に，スケジューラの導入による階層構造化，ER（実体一関連）図への変換がある.

6. 適用範囲と応用

ソフトウェア開発技法には汎用の技法はないというのがジャクソンの主張である. 技法は課題の解き方である. 汎用の課題があるとすればそれの解き方が汎用の技法である. 今のところソフトウェア開発において汎用の課題はないし今後もないであろう．JSD にも適用できる範囲がありそれが明確である. 解ける課題か解けない課題を明示的に示し, 適用に失敗したことがはっきり分かる技法が推奨できる技法と言える.

6.1 JSD の限界

(1) 静的側面のみを対象とした場合に余分な作業をさせるという弱点を持つ.

(2) 以下は適用対象外である.
　ネットワーク設計, ソフトウェア構成法, データベース設計，プロジェクト管理.

(3)アルゴリズムの設計が課題である場合は適さない.

(4) 実体の構造が変化するような課題領域が変化するシステムには適さない.

(5) システムの要求定義やシステムのアーキテクチャの設計では適用できない．JSD はソフトウェアの開発方法である.

6.2 JSD の応用例

JSDを応用しているMERODE（Ｍｏｄｅｌ driven Entity-Relationship Object oriented Development）[15]では，クラスの抽出と定義においてはUse-Caseのアプローチよりも優れているという理由で, JSDをオブジェクト指向におけるクラスの抽出と定義で採用している.

JSDの入力プロセスで文脈検査などモデルプロセスを状態ベクトル参照する機能は３層モデルではプレゼンテーション層に入れないことが多い. JSDの実現における手法を使い入力プロセスを分割することで, 入力プロセスはプレゼンテーション層と機能層に, 情報機能と会話機能プロセスは機能層, モデルプロセスはリソース層というように仕様書を3層モデルの層に配置できる.

MERODEとJSDモデルとの対応は図14でFowler，MERODE，JSDとの対応関係を図15で示している．これらから，JSDは計算モデルが異なるが, 変換によってオブジェクト指向のクラス設計まで応用できることが分かる.

図14 MERODEとJSD

図15 Fowlerのモデル，:MERODEとJSDのモデルの関係

7. まとめ

ジャクソンは“ソフトウェア開発方法とは, 次のような規約を持つひとまとまりの記述を製作するための製作計画であり, 記述の正当性を保証するための手段と規則も含む.”と述べている.

・何を記述するのか
・記述の順序は何か
・製造の作業と記述の作成に使うツールは何か
・使う言語は何か

このような規約を満たすソフトウェア開発技法は少ない. JSP と JSD はこれらを満たしている. 次に JSD が持っている課題と応用にあたっての注意をまとめておく.

・ JSD における実体の属性は機能の仕様化を通じて追加される. 実体の属性をすべて記述し尽くしているわけではないし尽くせない.
・ 実世界が実体/役割ですべて記述されているわけではない.
・ 出来事の粒度が実体/役割の粒度を決定する.
・ 属性のデータ型で抽象データ型を扱っても良い. このことは JSD 以外の技法で抽象データ型を定義することを意味している. JSD では解決しない部分であり技法の組み合わせの必要性を示している.

JSD は 20 年前に開発された技法である. しかしその内容は洞察に富み, 年代の技術に応じた適用方法があると考える.

参考文献

[1] M.jackson：Principles of Program Design , Acdemic Press 1975

[2] M.jackson：System Development,Prenice Hall, 1982

[3] J.Cameron, et al：Tutorial JSP&JSD, The Jackson approach to Software Development, IEEE 1989

[4]J.Cameron：An overview of JSD, IEEE Trans. of SE Vol SE-12 No 2, Feb. 1986

[5] J.Hughes：A Formalization and Explication of the Michael Jackson Method of Program Design, Software-Practice and Experience, Vol. 9, 1979

[6] D.McCracken, M.Jackson：Life Cycle Concept Considered Harmful, ACM SIGSOFT Vol.7 No.2, April 1982

[7] C.Ramamoorthy, et al.：Programming in The Large, IEEE Trans. on SE, Vol.SE-12 No.7, 1986

[8] 加藤：JSD (Jackson Syatem Development) 仕様の実行系の試作,「プロトタイピングと要求仕様」シンポジウム (1986 年 4 月 16 日) 情報処理学会

[9] J.kato, Y.Morisawa：Direct Execution of a JSD specification Proc. of COMPSAC87, 0ct.1987

[10] 大野:ジャクソンシステム開発法, 情報処理学会, 情報処理 Vol . 25 NO.9 Sep. 1984

[11] C.Hoare：Comunicasing Sequential Process, CACM Vol.21 No.8, 1978

[12] A.Goldberg：SMALLTALK-80 The interactive Programming Environment Addison-Wesley 1984

[13] P. Ward, S. Mellor Structured Development for Real-Time Systems Yourdon Press 1985

[14] D.Coleman, et al Object-Oriented Development: The Fusion Method Prentice-Hall 1994

[15] M.Snoeck, et al Experiences with Object Oriented Model-Driven Development Proc. STEP'97 1997

[16] M.Fowler Analysis Patterns Addison Wesley 1997

[17] P.Chen The Entity-Relationship Model – Toward a Unified View of Data ACM Trans. On Database Systems Vol.1 No.1 1976