TM-0989

代数的仕様と時制論理による リアルタイムSAの形式的支援

本位田　真一、大須賀　昭彦、
内平　直志（東芝）

December, 1990

ICOT
Mita Kokusai Bldg. 21F
4-28 Mita 1-Chome
Minato-ku Tokyo 108 Japan
(03)3456-3191～5
Telex ICOT J32964

Institute for New Generation Computer Technology

ソフトウェア工学 73－3
（１９９０．７．１９）

代数的仕様と時制論理によるリアルタイムＳＡの形式的支援

本位田真一　大須賀昭彦　内平直志

（株）東芝　システム・ソフトウェア技術研究所

リアルタイム・システムを対象として、コンテクスト・ダイアグラムからデータフローダイアグラム（ＤＦＤ）として機能展開し、Ａｄａのタスクを生成するまでの設計プロセスを明らかにする。その際にＤＦＤのバブルを代数的仕様で記述し、その記述内容を項書換え系で検証する。さらに、ＤＦＤのストアを多重アクセスデータとして、それに対する演算のアクセス順序を示す状態遷移図を時制論理仕様から定理証明系によって生成している。

A Formal Method for Real-Time SA using Algebraic and Temporal Logic Specifications

Shinichi Honiden　Akihiko Ohsuga　Naoshi Uchihira

Systems and Software Engineering Laboratory, Toshiba Corporation
70 Yanagi-cho, Saiwai-ku, Kawasaki 210, Japan

This paper describes a formal method for Real-Time SA using algebraic and temporal logic specifications. In data-flow diagram, "bubble" is described by algebraic specification, is verified by term rewriting system, and access sequence for "store" is synthesized from temporal logic specification by theorem prover. We also determine the concrete specification process from context diagram to generate concurrent program written in Ada.

1．はじめに

種々のソフトウェアの仕様化手法の中で形式的手法はそれによって得られるソフトウェアの品質や信頼性を高める手法として広く認められている。しかし、リアルタイムシステムを対象とした場合、産業界では充分に活用されているとは言い難いのが現状である。形式的仕様が実用に供せられるためには、少なくとも次の要件に対して何らかの形で応ずる必要があると考える。まず第一に所望のシステムの性質をできる限り多く記述するために複数の形式的手法を組み合わせることである。これは、ひとつの形式的手法で所望のシステムの性質をすべて記述するのは非常に困難であることによる。たとえば、代数的仕様はデータの抽象化や型定義には適しているが、並行性，同期，通信の記述には向いてはいないものが多い。また、時制論理仕様はその逆である。第二としては、形式的手法を用いてどのようにabstract specificationからdetailed specificationに展開するかといったspecification refinement processを確立させることである。別の言い方では設計方法論である。これは、経験の少ない人にとっても問題なくspecification refinement processを進むことができることの意義は極めて大であることによる。また、形式的な記述に習熟するにはかなりの訓練と時間を必要とする。さらに、複数の人に同一のabstract specificationを与えた時、途中のspecification refinement processおよび得られるソフトウェアにおいて、できる限り差異が生じないようにすることも必要である。第三には、formal method での記述内容を簡便な手法で検証・確認する手段が必要である。記述された仕様のバグを容易に見つけることのできる手段も合わせて必要である。第四には、formal method を用いて記述していると、どうしても細かい部分に焦点が移るが、対象とするソフトウェアの規模が大きい場合には全体を見渡すことのできる手段も必要である。第五にformal method で記述されたspecification から、いかに自然にprogram を生成するかである。いかに厳密にspecification を記述できたとしてもそれがprogram に直接結びつかないのであるならば、実用的な利用は困難であるという結果になる。これらの要求項目に応ずるために従来より多くの仕事がある。まず第一の複数の形式的仕様を効果的に組み合わせる手法については［Vautherin87］［Kramer87］など多くの提案がある。しかしながら、それらの提案が第二～第五までの要求をすべて満たしているとは言い難い。そこで本論文では、並行プロセスを有するリアルタイムシステムを対象としてこれらの要求項目に対してどのように対処するかについて述べる。

まず第一に複数の形式的手法として代数的仕様と時制論理の組み合わせを採用する。代数的仕様によって静的な側面である機能やデータに関する性質を記述し、時制論理によって動的な側面である並行性，同期，排他に関する性質を記述するものである。

第二に設計方法論としてはリアルタイムシステムにおいて広く利用されているリアルタイムＳＡ(Structured Analysis) を採用する。リアルタイムＳＡにおいてはＤＦＤ(Data-Flow Diagram) におけるバブルの記述に代数的仕様を適用し、制御スペックの記述に時制論理を適用する。著者らは、ＤＦＤにおけるバブルの記述に代数的仕様を適用した結果としてバブルの機能展開基準が明確になり、きめの細かい設計プロセスが規定できることを既に示している［古川90］。きめの細かい設計プロセスを規定することにより、得られるソフトウェアにおいて個人差を少なくすることも可能になる。

第三の課題に対しては、代数的仕様，時制論理には様々な体系が存在するが、できる限り自動検証可能な範囲に限定させている。また、代数的仕様による記述内容をできる限りＤＦＤに変換することにより、視認によるvalidationも合わせて可能にしている。検証内容によっては種々の検証項目による機械的な自動検証よりも視認の方が効率的な場合が多いからである。

第四の課題に対しては、ＤＦＤでの階層関係に従って、上位のＤＦＤから下位のＤＦＤへ、また

はその逆というように自由に記述世界をtraverseできるようにし、全体を見渡すことを可能とする。

第五に対しては、ＯＯＤ(Object-Oriented Design) の概念を導入する。具体的には、上位のＤＦＤに現れるストアをオブジェクトの保持する内部状態データとする。さらにストアの回りの各バブルを下位のＤＦＤに機能展開することにより、当該オブジェクトのメソッド部分を切り出す。すなわち、データに必要なメソッドを洗い出すことによりオブジェクトを構成できることになる。

さらに、メソッド間の実行の順序関係を制御スペックである状態遷移図で表わす。この状態遷移図は、時制命題論理から定理証明系によって生成される。このデータ，メソッド，および状態遷移図から並行プログラムを生成する。

以上の様に形式的手法を実用に供するための種々の課題に対する解決案を示すのが、本論文の目的である。

２．概要

本章では、提案手法の概略とそれを支える要素技術について述べる。

まず、仕様記述の核となっているリアルタイムＳＡを明らかにする。本手法で扱うリアルタイムＳＡ手法は、DeMarco のＳＡをリアルタイム向けに拡張しているが、その拡張部分はストアに対する複数のバブルからのアクセスにおいて、そのアクセス順序を状態遷移図によって記述することである。従ってWardやHatleyのリアルタイムＳＡをすべてカバーしているわけではない。Hatley流で言えば、状態遷移図で記述する制御スペックの対象範囲をストアの回りのアクセス順序に限定している。

入力仕様はリアルタイムＳＡのコンテクスト・ダイアグラムであり、出力形態はＡｄａのタスクである。提案手法は次の５つのphase から構成される。

phase1：ＤＦＤを展開することによるデータオブジェクトの生成［古川90］
phase2：データ・オブジェクトの実現レベルでの記述
phase3：項書換え系による検証
phase4：時制論理による制御スペックの生成
phase5：Ａｄａのタスクの生成

<phase1>

まず、与えられた仕様のコンテクスト・ダイアグラムを記述することから始まるが、このコンテクスト・ダイアグラムは単一のオブジェクトに相当する。このオブジェクトにおける入出力データ（これが対象システムの外界とのインタフェースを規定）に対して代数的仕様を用いて記述する。次に、このコンテクスト・ダイアグラムをＤＦＤに展開する。この時各バブルの機能を代数的仕様によって記述する。さらにＤＦＤの各バブルを終了条件を満たすまで展開基準に従って展開していくことを繰り返す。本手法では展開基準と終了条件を次の様に規定している。

（展開ルール１）

出力データ一つに対して機能が一つ対応するように分割する。（一般に代数的仕様では機能に対応する演算は副作用のない関数の形で定義されるため）

（展開ルール２）

等式の右辺の記述に基づき展開する。バブルにおいて等式表現によってそのバブルの機能の性質を表現する。この等式の右辺に記述されている式は、機能の働きを表現していると同時に機能の展開を表しているとみなすことができる。

（展開ルール３）

ストアはデータオブジェクトとしてまとめる。

（終了条件）

バブル内の等式表現の右辺がprimitive functionあるいはrecursive functionで構成された時。

phase1では、上の終了条件を満たすまで上の展開ルールによって機能展開を繰り返しながら最終的な出力であるＡｄａのタスクの骨格を作っていく。その手順は以下の通りである。

ＤＦＤで現れるストアを中心として展開ルール

3に従いデータオブジェクトとしてまとめる。このデータオブジェクトが生成すべきタスクの基礎となる。ストアの回りのバブルにおいて機能展開を繰り返すことによって、ストアと関係が深いfunction、言い換えれば、ストアを直接アクセスするfunctionを洗い出す。洗い出したfunction群はデータオブジェクトに登録される。等式があるデータオブジェクトに宣言されたデータを直接アクセス（参照，書込み，更新）するならば、そのデータオブジェクトにシステムが自動的にデータをアクセスするfunctionとして登録する。このようにしてデータオブジェクトのデータに関わるfunctionを抽出していく。

<phase2>

phaselにおいてＤＦＤを展開し、データ・オブジェクトを作成したが、この中での等式記述はあくまでも外部仕様レベルである。言い換えれば、データ・オブジェクト内の物理データ構造を意識しない記述である。phase2においてはデータ・オブジェクトが含むデータの物理データ構造をまず決定する。その際には、対象とするアプリケーションを意識することも必要である。phaselで生成したデータオブジェクトにおいて登録されているfunctionはデータの物理的構造や実現レベルの内容を含んでいない。その理由は、それらのfunctionはデータオブジェクトの外側からデータを把えており、抽象データ型に対するfunctionであるとみなすこともできるためである。そこでphase2ではデータオブジェクトに対しては等式を用いて物理的データ構造を踏まえて詳細に記述する。phaselまでの記述内容はfunctionに対する外部仕様であり、一方phase2の記述は内部仕様であるとみなすことができる。

<phase3>

phase2で記述した等式による詳細な記述内容を項書換え系によって検証する。検証するためには、その記述内容に対して意味を与える必要がある。抽象データ型の代数的仕様の意味定義法としては始代数(initial algebra) や終代数(final algebra)などがあるが、記述内容の性質を調べる手続きが不十分である。そこで代数的仕様の計算モデルとして手続きが比較的整備されている項書換え系を採用する。項書換え系は等式で記述された仕様に対して操作的な意味を与え、等式を左辺から右辺への書換え規則とみなして計算を行うモデルである。この項書換え系においては、与えられた項書換え規則の集合が停止性，合流性を満たすように新たな書き換え規則を生成する手続き（完備化手続き）が存在する。この手続きによって代数的仕様の無矛盾の検証が可能となる。ここでの検証項目は大きく２つである。ひとつは項書換え系の完備化手続きによって停止性，合流性を保証するものである。もうひとつは、ユーザが任意の検証項目を等式で与えることにより、データオブジェクトの記述内容（等式）が検証項目を満たしているかどうかの吟味に相当する。いずれの場合においても完備化手続きが止まらない場合がある。その場合には検証不能である。従って検証可能範囲は、完備化手続きが停止する場合に限定される。

<phase4>

検証済みのデータオブジェクトにおいては、複数のfunctionからの多重アクセスが発生する。生成するのはＡｄａのタスクであるため、モニタ機構によって、あるfunctionの実行中には他のfunctionの実行はsuspend される。しかし、function間にまたがる排他制御（たとえば、あるfunction "a" の実行後にはfunction "b" を実行するまでfunction "c" を実行してはならない）に対してはこのままでは対処できない。そこでfunction間の実行順序を制御スペックである状態遷移図で記述する。リアルタイムＳＡの制御フローは、状態遷移図である。この状態遷移図は、人手で作成するために正当性は保証されていない。本手法ではWolperらの手法を採用する。すなわち、Ａｄａにおける内部データに対するfunctionのアクセス順序を時制論理で記述した仕様から定理証明系によって生成するものである。以下に詳細を示す。Wolperの手法はタブロー法に基づいている。タブロー法とは与えられた論理式が恒真であるか

を証明するために、状態遷移図を生成する。状態遷移図においてはアーク上の論理式はアークの元にあるノードにおいて真となることを示している。従って論理式に何を割付けるかによって状態遷移図の解釈が異なってくる。ここでは、論理式に対して、データをアクセスするfunctionをアサインする。それにより状態遷移図はfunctionの実行順序を示していることになる。リアルタイムシステムにおいては、イベントの系列である制御フロー以外に時間制約(timing constraint)の記述が必要である。さらにリアルタイムＳＡで扱う制御スペックには種々の条件式が現われている。従って本手法においてもこれらの２つの記述を支援する必要がある。そこで、時間制約に関しては[Dasarathy85]における記述を採用し[Honiden90]、条件式は状態遷移図上のアークに付加する。

<phase5>

データ・オブジェクトにおけるデータとそれに関わるfunctionおよびphase4で作成した状態遷移図からＡｄａのタスクを生成する。ＤＦＤのストアはこのタスクに情報隠ぺいされている内部データとなり、ストアに対する各functionはタスクの各accept文となる。すなわち、複数の外部からの呼出し(entry-call)に対して、条件付きのaccept文で待機している形態をとる。when文の後の条件式の表わす数値は状態遷移図のノード番号を示す。また、等式表現はＡｄａのfunction形式に変換される。

3．評価

提案手法ではリアルタイムＳＡをベースとしてデータフローは代数的仕様，制御フローは時制論理によって記述している。

一般にリアルタイムＳＡの持つ問題点として

①バブルの記述があいまいである。（形式的ではない）

②バブルの厳密な検証手段がない。

③機能展開基準がない。

④展開の終了条件がない。

⑤下流工程とのつなぎが現実的でない。（タスクの生成手順が不明確である。）

⑥制御スペック（状態遷移図）の正当性が保証されていない。

⑦設計プロセスが必ずしも明確ではない。

などがある。これに対して本手法では、

①' 代数的仕様記述言語を採用する。

②' 項書き換え系を採用する。

③' バブルの出力がひとつになるように展開する（等式記述の制約による）。さらに等式記述の右辺に基づいて展開する。

④' recursive functionあるいはprimitive function（登録済みのfunction）で等式記述の右辺が構成された時。

⑤' ストアをＡｄａのタスクの内部データとして、それに対するfunctionをストアの回りのバブルの機能展開の過程で抽出していくことによりＡｄａのタスクを生成する。

⑥' 時制命題論理からその定理証明系であるタブロー法によって状態遷移図を生成する。

⑦' 代数的仕様とデータフローダイアグラムを融合し、制御スペックをタスク内のfunctionの実行順序であるとすることにより設計プロセスを規定している。

また、抽象データ型の設計プロセスとしては一般に、

① 抽象データ型を外部からアクセスするfunctionを定義する。（この時は、外部のfunctionは抽象データ型の実現レベルについては知る必要はない）

② ①の方法で抽象データ型に対するfunctionを洗い出す。

③ 抽象データ型の内部では、具体的な物理的データ構造を踏まえた実現レベルでのfunctionを定義する。

④ 抽象データ型から具体的なプログラムに変換する。

といった流れになるが、本手法での設計プロセスはこの流れに沿ったものとなっている。

本手法の制約として次の項目があげられる。

① バブルの出力をひとつとしていること。
② 制御スペックの記述をタスク内のfunctionの順序関係に限定していること。
③ ストアをタスクの核としているが、ストアのないタスクに対しては特に支援していないこと。

検証系からみた制約としては代数的仕様に関しては検証する際の完備化手続きが停止しないことがある。この時には検証不能となる。一方時制論理仕様に対しては、正当性が保証されているのは時制命題論理から生成される状態遷移図のみであり、付加された条件式は検証の範囲外である。また、時間制約に関しては、それを満たすように状態遷移図を縮約させているが、個々のfunctionの実行時間がfix していることを仮定している。

本システムは現在実装中であるが、ＤＦＤを表示する window と代数的仕様によって記述するwindowがある。この２つのwindowの情報は一貫性を保っており、ユーザがどちらかを修正すればシステムが自動的に片方のwindowの表示を修正することになる。また、ストアや中間データ，外部との入出力データの詳細を記述するwindowがあり、ストアの回りのバブルを展開することによって得られた等式による記述内容のなかで、ストアを直接アクセスするfunctionはシステムによって自動的にこのwindowに登録される。形式的仕様記述を実レベルに適用する場合、しかも一般ユーザに利用させるには支援環境の充実が必須である。この観点で述べるならばきめの細かい設計プロセスを支援環境が誘導(navigate)しており、図式仕様記述との併用も、ユーザにとっては有効である。

4．他の研究との比較

本章では次の６点から他の研究との比較を行なう。形式的アプローチの統合化手法，等式論理＋ＳＡ，設計方法論，設計プロセス，ＯＯＤと並行プログラム，支援環境，である。まず、形式的アプローチの統合化手法としては[Vautherin87]，[Kramer87]などがある。いずれの手法においても，機能・データの記述にＶＤＭや代数を用いており、同期，並行性の記述には、時制論理，ペトリネット，ＣＣＳ，ＣＳＰを用いている。このように、種々の手法が採用されているが、必ずしも検証手段が整備されているとは言い難い。またＬＯＴＯＳは言語として両面をサポートしているが検証系については今後の課題である［大蒔90］。本手法は機能，データの記述には代数的仕様を用いており、同期部の記述には時制論理を採用している。システムとして実装するのが目的であるため、あくまでも解析可能範囲，言い換えれば機械的に検証できる範囲の記述力にとどめている。そのため、時制論理においては、領域計算量がＰＳＰＡＣＥ完全である時制命題論理を採用している。代数的仕様においては、項書換え系で検証可能な範囲に限定している。また、形式的仕様記述の読解性を向上させるために、図式仕様記述手法としてデータフローダイアグラムを採用している。これにより記述内容を視認することも可能としている。

[Docker89]は等式論理とデータフローダイアグラムを融合している。[Docker89]との相違は、本手法は代数的仕様とデータフローダイアグラムを融合することにより、設計プロセスを規定していることである。

リアルタイムシステム向けの設計方法論としてはＪＳＤ[Cameron89]，ＤＡＲＴＳ[Gomaa86]，リアルタイムＳＡ／ＳＤ[Ward86,Hatley84] などがある。各々の設計方法論を分類する方法には種々あるが、大きな項目としてはまず何からモデル化するか（抽出するか）があげられる。リアルタイムＳＡ／ＳＤではデータフローダイアグラムを記述することによりデータフロー解析から始まる。本手法もリアルタイムＳＡをベースとしているので、必然的にデータフロー解析をまず行う。ＤＡＲＴＳはＳＡ／ＳＤを拡張したものであるから、やはりデータフロー解析から始まる。それに対してＪＳＤではデータフロー解析ではなく、アクションおよびそのsequenceの抽出が最重要項目である。また、データとイベントをＳＡ／ＳＤやＤＡＲＴＳは明確に分割しているのに対してＪＳＤは分割していない。本手法においてはデータを代数的仕

様記述でイベントを時制論理で記述することにより、明確に分割している。

設計方法論としての比較の最後として、設計プロセスを経てどのようにプログラムを生成するかである。ＪＳＤは設計プロセスの中にimplementation stageが存在する。ＤＡＲＴＳではデータフローダイアグラムからいくつかの基準（たとえば同じstimulusによってtriggerされるtransformationは同一タスクにする）によってタスク分割を行っている。リアルタイムＳＤにおいては、central transformationの考え型でモジュール構造を実現している。本手法ではＯＯＤの考え型を採用してデータとそれに関わるfunctionをオブジェクトして抽出している。WardはリアルタイムＳＡとＯＯＤの関係について論じた[Ward89]。この中で、リアルタイムＳＡの設計方法論の枠組でＯＯＤを支援することが可能であると述べている。すなわち、最下位層のデータフローダイアグラムにおいてストアを内部データとし、ストアを中心として回りのバブルをオペレーションとすることによってオブジェクトとみなそうとするものである。本手法とは近いが、本手法では機能展開を繰り返すことによってオペレーションを抽出していく点が異なっている。また、制御スペックの取り扱いについては特に触れていない。また、ShlaerとMellorはリアルタイムＳＡ[Hatley84]を発展させたＯＯＡ(Object-Oriented Systems Analysis) を提案している[Shlaer87]。ＯＯＡではまずオブジェクトを抽出し、そのオブジェクト自身の状態の変化を状態遷移図で表現し、各状態における機能をデータフローダイアグラムで記述するものである。Wardのリアルタイムと ＯＯＤの結合手法では最後にオブジェクトを抽出するのに対して、ＯＯＡではまずオブジェクトを抽出している。本手法はWardのアプローチと近い。また、CoadもＥＲダイアグラムとＤＦＤをベースとしたＯＯＡについて提案している[Coad90]が、まずオブジェクト抽出から始まる。いずれにしても、ＪＳＤ，ＤＡＲＴＳ，Ｗａｒｄの手法，ＯＯＡは形式的手法ではないことが本手法とは根本的に異なる。

設計プロセス自体をプログラミング言語で書く下すプロセスプログラミングには数多くの研究がある。しかし、これらの研究の対象としている設計プロセスは本手法で扱っている設計プロセスよりも粗いものである。具体的には中間生成物としては、それらの研究では外部仕様書，プログラム，テスト仕様などであるのに対して、本手法では外部仕様書，内部仕様書を作り出すプロセスにおいて生成される中間生成物を対象としているのが大きな相違点である。

ＯＯＤを用いてＡｄａのタスクを生成する研究にはＯＯＳＤ[Wasserman90]，ＨＯＯＤ[Hood89]，ＯＯＤ[Booch86] などがある。これらの研究では、リアルタイムＳＡやＳＡからの連続的な展開ではないこと、および生成されるＡｄａのタスクにおいてprocedure 間の排他問題を扱っていないことが本手法との相違点である。連続的な展開の観点ではＪＳＤからＡｄａプログラムを生成する研究がある[Cameron89] 。これはＪＳＤにおけるネットワーク仕様(network specification) から素直にＡｄａのタスクを生成するものであるが、タスク内のprocedure 間の順序関係は規定していない。さらにＪＳＤでは時間制約を扱っていない。

また、形式的仕様記述支援ツールの点においては、構文エディタ，仕様ライブラリ，デバッガ，検証条件生成系，直接実行系など数多くの支援ツールに関する研究がある。本手法で提供している支援ツールの中で、他にはないツールとしては代数的仕様とＤＦＤの両方向一貫性支援ツールおよびＤＦＤの機能展開に関する誘導（ナビゲータ）の２つをあげることができる。

５．むすび

本論文では、形式的仕様記述を実レベルの問題に適用する際のいくつかの課題を解決することを試みた。その時のポイントは

①時制論理と代数的仕様を組み合わせること。

②リアルタイムＳＡを採用したこと。

③きめの細かい設計プロセスを規定したこと。

④ＯＯＤの考え方により並行プログラムを生成したこと。

である。今後の課題としては、まず第１として、きめの細かい設計プロセスにおける中間生成物の形式化においては見通しを得ることができたが、作業項目の形式化，知識化は未解決である。作業項目も知識化できれば、自動プログラミングへ一歩大きく近づくことになる。また、設計プロセス自身の再利用も可能になる。もうひとつの課題としては、実際のシステムに適用することにより、人間の知的活動が必要な部分と計算機で対処可能な部分を明確に分類することにある。

形式的仕様記述の支援環境としては効率的なデバッキング・ツール，仕様ライブラリなど充実しなければならない項目が残っている。

本研究の一部は第５世代コンピュータプロジェクトの一環として行なわれた。

参考文献

[Booch86] G.Booch, Object-Oriented Development, IEEE Trans. Software Eng., Vol.SE-12,No.12,1986

[Cameron89] J.Cameron, JSP&JSD: The Jackson Approach to Software Development, IEEE Computer Sciety, 1989

[Coad89] P.Coad, E.Yourdon, Object-Oriented Analysis, Yourdon Press, 1990

[Dasarathy85] B.Dasarathy, Timing Constraints of Real-Time Systems: Constructs for Expressing Them, Methods of Validating them, IEEE Trans. Software Eng., Vol.11,No.1,1985

[Docker89] T.W.G.Docker, Flexibility and Rigour in Structured Analysis, IFIP'89, 1989

[古川90] 古川，本位田，大須賀，津田，代数的仕様記述と図式仕様記述の相補的役割について，情報処理学会論文誌，Vol.31,No.2,1990

[Gomaa84] H.Gomaa, A Software Method for Real-Time Systems, Comm. of the ACM, Vol.27,No.9,1984

[Hatley84] D.Hatley, The Use of Structured Methods in the Development of Large Software Based Avionics Systems, AIAA/IEEE sixth Digital Avionics Systems Conf., 1984

[Honiden90] S.Honiden et al., An Application of Structural Modeling and Automated Reasoning to Real-Time Systems Disign, The Journal of Real-Time Systems, Vol.1, No.3,1990

[Hood89] Hood Working Group, Hood Reference Manual, European Space Agency, 1989

[Kramer87] B.Kramer, SEGRAS-A Formal and Semigraphical Language combining Petri Nets and Abstract Data Types for the Specification of Distributed Systems, 9th ICSE, 1987

[大蒔90] 大蒔：private communication

[Shlaer88] S.Shlaer, S.J.Mellor, Object-Oriented Systems Analysis, Prentice-Hall, 1988

[Vautherin87] J.Vautherin, Parallel Systems Specifications with Coloured Petri Nets and Algebraic Specifications, LNCS 266, 1987

[Ward86] P.Ward, The Transformation Schema: An Extension of the Data Flow Diagram to Represent Control and Timing, IEEE Trans. Software Eng., Vol.12,No.2,1986

[Ward89] P.Ward, How to Integrate Object Orientation with Structured Analysis and Design, IEEE Software, Vol.6,No.2,1989

[Wasserman90] A.I.Wasserman et al., The Object-Oriented Structured Design Notation for Software Design Representation, IEEE Computer, Vol.23, No.3,1990