ソフトウェア工学 92－1
（1993．5．26）

# リアルタイム性を追求した オブジェクト思考設計法ＯＣＡ

(Object modeling Consideration Analysis:OCA)

龍忠光　泉寛幸　村川雅彦

株式会社 富士通研究所

SA/SD, IMM, OMT, OOA/OOD 等いろいろなシステム設計法が提案されてきたが，部品化やインプリメント等においていまだ充分考慮されてきたとはいいがたい。我々は，ＯＣＡ (Object modeling Consideration Analysis) と呼ぶシステム設計法を考案し，その設計法に基づき，プログラムの再利用に着目したリアルタイム・アプリケーション向けシステム構築環境を開発している。本報告においては，システム設計法ＯＣＡにおける世界のモデリングの方法と部品化のやり方，さらにリアルタイム・アプリケーションで必要なシステム内で並列に動くイベント間の同期（因果関係）のとり方について報告する。

# OCA:Object modeling Consideration Analysis for Real Time Applications

Tadamitsu Ryu　Hiroyuki Izumi　Masahiko Murakawa

FUJITSU LABORATORIES LTD.

Various system design methods have been proposed, but none of them considered the enough support for the modularization of programs and the implementations etc. We have studied one design method OCA (Object modeling Consideration Analysis), and developed the programming environment for the realtime applications with the reusability of the software based on it. This paper discusses OCA, its software modularization and moreover, its synchronization between the events which run concurrently in the realtime applications.

## 1. はじめに

従来のオブジェクト指向は，ソフトシステムの構築において，プログラムの部品化，システム構築の柔軟性，保守の容易性等に対する充分な機能をもっていない。そのため，SA/SD, IMM, OMT, OOA/OOD 等いろいろなシステム設計法が提案されてきた。SA/SD[1]は手続き型プログラミングに合わせてデータフロー図やデータ辞書等を作成していくがオブジェクト指向に基づいた設計はできない。IMM[2]はE-R モデルをオブジェクト指向をもとにデータベース分野からまとめたものであり，静的なオブジェクトモデルのみ考慮されており，動的な記述が明確でない。OOA[3]においても同じである。OMT[4]は実世界をオブジェクトモデル，動的モデル，機能モデルで表現している。しかし実際のインプリメント法が考慮されていない。

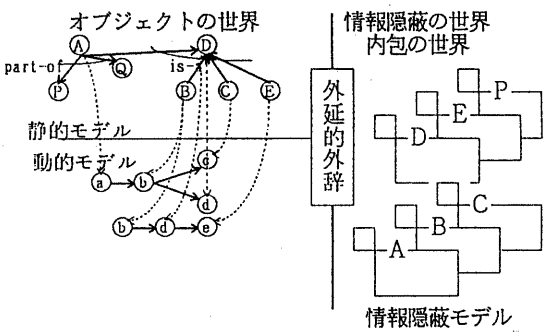
我々はOMT を改良し，実世界を情報隠蔽の世界とオブジェクトの世界とに分け，さらにオブジェクト世界を静的モデルと動的モデルとに分けて記述し，実世界の因果関係をオブジェクト世界で記述することにより，実際のインプリメントをも考慮したモデリング手法の開発を試みている。このモデリング手法を“従来のオブジェクト指向にこだわらずオブジェクト的発想をするオブジェクト思考”という意味で「オブジェクト思考設計法」OCA(Object modeling Consideration Analysis)と呼んでいる。以後，従来のオブジェクト指向との混同を避ける意味でオブジェクト思考設計法をＯＣＡと略記する。

本報告においては，村川[12]の発表に従って，このＯＣＡの概観を２節にまとめ，３節においてＯＣＡにおける部品化の考え方，４節においてリアルタイム性を実現するための機構として因果関係の動かし方について述べる。

## 2. オブジェクト思考設計法（ＯＣＡ）

### 2.1 実世界のモデル化

ＯＣＡでは，実世界の対象（オブジェクト）を外延と内包とに分けてとらえる。我々は，ものごとの性質や関係を考える場合に，もの自身でなくものを示している名称，すなわち，外延的に理解している名称（便宜上，外延的外辞と呼ぶ）を用いて，その関係や性質を考える。そして，外延的外辞の示すものの性質や内容をより深く考えるときのみ内包（外延の持つ内容や性質）にたちいって考える。ＯＣＡではこの外延的外辞と内包との考えに従い，実世界をまず，外延的外辞の世界と内包の世界とに分けて考える。内包の世界を「情報隠蔽の世界」と呼ぶ。外延的外辞の世界はさらに「静的世界」と「動的世界」とに分ける。そして，モデル化の対象をこの３つの世界での側面に従ってモデル化することにし，静的モデル，動的モデル，情報隠蔽モデルの３つを構成する。




第１図　実世界のモデル化

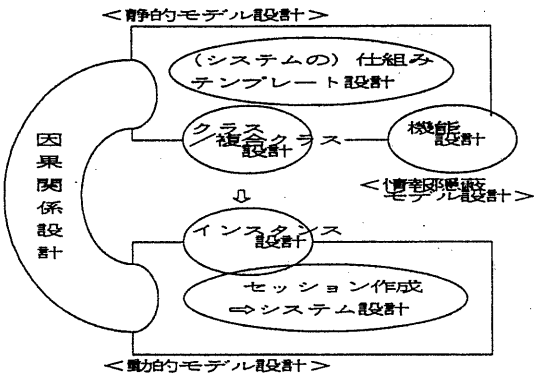
「静的モデル」とは，モデル化の対象の”時間をとめても同時に存在する関係”を示すものであって，クラスやクラスを組み合わせた複合クラス，さらにはそれらをテンプレートに対応させたもので表現される。このクラスは，オブジェクト指向におけるクラスの属性を「属性の振る舞い」と「属性の状態値」とに拡張したものとし，属性の追加，削除を自動的に行ったりするような空間的な推移によるオブジェクトの状態変化に対応できるものとした。

「動的モデル」とは実現しようとしている動きについて示すモデルであり，”時間を止めると同時には存在しない関係”である。動的モデルは，静的モデルで定義したクラスに対して実際に属性値を与えたインスタンス，インスタンスの組合せに

より小さな機能を持ったセッション，セッションの組合せによりユーザ仕様レベルとするシステム等で表現される。セッションの組合せを後述の因果関係により制御することによって並列動作をも表現する。

「情報隠蔽モデル」とは，外延の示す性質や内容を表現しているモジュール，およびそれらを組み合わせた複合モジュールであり，「情報隠蔽の世界」内に格納されている。外延の世界からは隠蔽されているが対応する外延とのインタフェースを持ち，必要に応じて参照される。

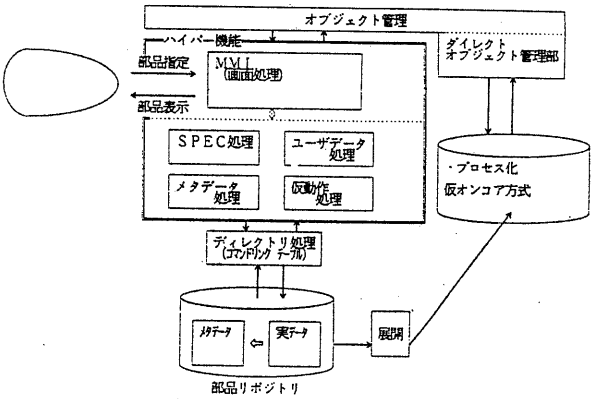
本モデルの設計段階において，インスタンスの動作間，セッションの動作間で，フラグ等の共通に用いられる資源の持つ条件や性質を，インスタンスやセッションとは別に定義する。この定義によって定められるインスタンスやセッション間の動作関係を「因果関係」と呼ぶ。動的モデル上のインスタンスやセッションが実際に動くときには静的モデル上の仕組みと，この因果関係とに従って動くことになる。




第2図　静的モデル，動的モデル

## 2.2　支援システム

我々は，このＯＣＡをサポートするシステムとして，ＦＩＮＥＳコントローラを開発している。ＦＩＮＥＳコントローラの機能は大きく２つに分けられる。一つはハイパー機能であり，もう一つはダイレクトオブジェクト処理機能である。この二つの機能を中心とした全体の機能構成図を図３に示す。




第３図　設計法の全体構成

### (1)　ハイパー機能

ハイパー機能によって，ユーザは自分の要求仕様に最適な部品を検索，選択し組み合わせてクラス設計，静的モデル，動的モデルを構築し，さらにはシミュレーションをも行える。ここでは簡単に述べるが詳細は[12]を参照されたい。ユーザはハイパー機能により次のような操作をおこなう。

①メソッド部品を組み合わせクラスを作成する。

②クラスに実際の属性値を与えて，インスタンスを作成する。

③インスタンスを組み合わせて，小さな機能を持つセッションを作成する。

④インスタンス間，セッション間の因果関係を記述する。

⑤仮動作機能によりシミュレーションを行う。

仮動作により満足な結果が得られれば，ハードウェア環境やアプリケーションのタイプに従って，ダイレクトオブジェクト機能による展開を行い，実稼働用のシステムとする。

### (2)　ダイレクトオブジェクト処理機能

ダイレクトオブジェクト処理機能は，仮動作機能により動作確認したシステムを実際に使用するハードウェア環境などに合わせて，展開処理を行い，従来の共通資源の生成，テーブル化，オブジェクトの動作から手続き型プログラムへの変換等

を行い，システムのリアルタイム性を高める機能をアプリケーション業種毎に備えたものである。ここでは詳細は述べない。

### 2.3 ＯＣＡの目的

ＯＣＡにおいては，プロトタイピング手法と仮動作（シミュレーション）により，性能よりも設計のしやすさとバックログをいかになくすかということに重点をおいている。そして，実際に必要な性能を出すことは，展開処理とダイレクトオブジェクト法という仕掛けによりおこなうことを考える。

#### 2.3.1 手続き型プログラミングの問題点

ＯＣＡは，従来のプログラミングの問題点を解決することを目標の一つとしている。今までの手続き型プログラムの難しさの原因および解決法を図４に示し，以下に説明する。

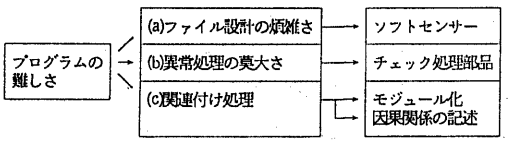



図４ 手続き型プログラミングの問題点

難しさの原因は３つある。

一つ目はファイル設計が煩雑であること，二つ目は異常処理の莫大さ，そして三つ目は共通資源（フラグ，共通レジスタ，共通ファイル）をあらかじめ厳密に設計しなおかなければならないことである。

#### 2.3.2 ＯＣＡの解決法

ＯＣＡでは，上記の問題に対して次のように問題の解決案を提案する。

(a)ファイル設計の煩雑さの解決策としては，ソフトセンサーと呼ぶ一種のファイルの自動化が行えるプログラム部品オブジェクトを用意する。詳細は，[8] を参照されたい。

(b)の異常処理の莫大さについては基本的なチェックルーチン（テーブルを入れ換えることでほとんどのチェックが動くという設計）を幾つか用意しておき，プログラム上に埋め込めるようにする。異常処理の例としては，正常シーケンスの条件に合わないときやデッドロックに陥ったときに，その内容をコメントとして異常終了したり，ロギングデータをとって初期状態に戻したりする操作がある。また例えば“月”を入れる場合１～１２の整数であるかどうかチェックするような入力データのチェックも異常処理に含める。

(c)の共通資源による関連付け処理においては，インスタンスやセッションを構成しているときには他のインスタンスやセッションとの関係を無視して設計する。そして，共通資源の管理はすべて因果関係のリンクにまかせるようにする。すなわち，共通資源の設計は，静的モデルや動的モデルや機能的モデルの上では考えない。それらの設計が終わってから，ソフトセンサーと呼ぶ変数を表すオブジェクト間に因果関係のリンクがはられるようにする。

## 3. 超部品

システム構築においては，トップダウン的にシステムの機能を分析し動作を設計していくのに，使われる部品はボトムアップ的に積み上げていく必要があった。このトップダウン設計法とボトムアップの部品合成法との二つの間を埋めるものがあればこれを超部品技術と呼ぶことにする。リアルタイムシステム構築の容易さをねらって，ＯＣＡは，この超部品技術の実現法を考える。

### 3.1 原子部品と複合部品

ＯＣＡでは，部品の再利用においては，原子部品と，複合部品との二つの概念がある。原子部品は基本単位となるメソッド部品で，複合部品はクラスやセッションのように原子部品や他の複合部品を組み合わせてつくるプログラム部品である。複合部品はその内包（定義内容）を情報隠蔽の世界に格納し，その外延的外辞でオブジェクト世界で参照される。原子部品は，少数の何でも使える部品であって，徹底的に単純化した小さなものと

しなければならない。そのためには，完全独立性であって，抽象化されており，正規化で冗長度をなくし，直交性で他の部品と組合せが可能であり，徹底的に単純化した小さなものとは，シリーズロジック，分岐のない系列的なものであろう。すなわち原子部品は，独立性，直交性，抽象性，簡易性，正規性をもつものとする。そして，原子部品を組み合わせて構成する複合部品にもこれらの性質を満たすように設計することを考える。超部品技術は，このような性質を満たす技術である。

(1)独立性とは，ある部品が他の部品と無関係に存在するようにすることである。

(2)直交性とは，他のどんな部品とも組み合わせることができる機能である。もちろん組合せた結果，意味の無い部品ができることもある。

(3)抽象性とは，応用分野を問わず，何にでも使えることである。

(4)簡易性とは，他の原子部品との組合せがシリーズロジック，分岐のない系列的なものであることである。

(5)正規性とは，部品として標準的なものが唯一できることである。

### 3.3 超部品化の仕掛け

超部品の上の５つの条件を満たすために，ＯＣＡは次のような仕掛けを用意する。まず，

(1)独立性を保つために，部品間の因果関係を部品とは独立に記述する。

(2)直交性を保つためにＦＩＮＥＳコントローラにより自動的に部品間の引数合わせが行なわれる。

(3)抽象性を保つために，データやメソッドをできるだけパラメータ化して超部品は格納され，使用時にそのパラメータが具体化される。

(4)簡易性のために，超部品を組合せるときには直列に並べて一つのセッションを作ることを中心に考える。分岐は一つのセッションから別のセッションに制御をうつすことと考え，その制御を行う分岐型コントロールＪＣＬをセッション内に組み込む。またループも一つのコントロールＪＣＬとしてセッション内に組み込む。

(5)正規性を保つために，プログラムのメタデータを表現しているシグネーチャやプログラム自身からできるだけ共通性を引き出して，is-a関係からなる超部品の階層化をすすめる。複合オブジェクトに対しては，そのオブジェクトを構成している部品のpart-of 関係の構造化をしておく。

超部品技術においては，セッションとして系列で並べたオブジェクト間の同期をとる因果関係が重要な役割を果たす。次節では，この因果関係について詳しくのべる。超部品技術の他のより詳細な点については，別稿にゆずる。

## 4. 因果関係

リアルタイム処理を行うためには，環境の急激な変化に対応するために，セッションの並列処理と異常時に対する異常処理とが不可欠である。そのようなリアルタイムシステムを構築するためには，並列に動くセッションおよびインスタンス間で同期をとる機構を各セッション毎に与えるよりも，個々のセッションはその同期を意識せずに作成し，同期機構を外部から与えるようにしたほうが構成が容易である。この外部から与える同期機構を，因果関係と呼ぶ。また，異常時に対応できるように，因果関係に従って特定セッションに動的に分岐する機構を強制分岐として実現する。

### 4.1 因果関係とは

因果関係は，並列に動くセッションオブジェクトに対してそれらの時間的な同期をとる制御である。実際にはセッション毎でなく，セッションを作っているインスタンス毎，その中のメソッド毎に同期をかける。

ＯＣＡにより構築したシステムを実行するときには，インスタンスやセッション等の間には，時間的，空間的，人為的，法則的な関係が存在するはずである。例えば図５において，ａ→ｂ→ｃというセッションｘと，ｄ→ｅ→ｆというセッションｙの２つでシステムを構築しているとした時，「インスタンスｂは，インスタンスｅが終了しなければ動作できない。」という条件があったとす

る。従来の手続き型プログラミングにおいては，両者の間に共通資源（フラグ）を導入する必要がある。ＯＣＡにおいては，このようなインスタンス間の因果関係を外部で設定することにより，各オブジェクトの独立性を維持することができ，かつ複数の並列的なセッションおよびインスタンスにおいて共有資源の概念を導入できる。

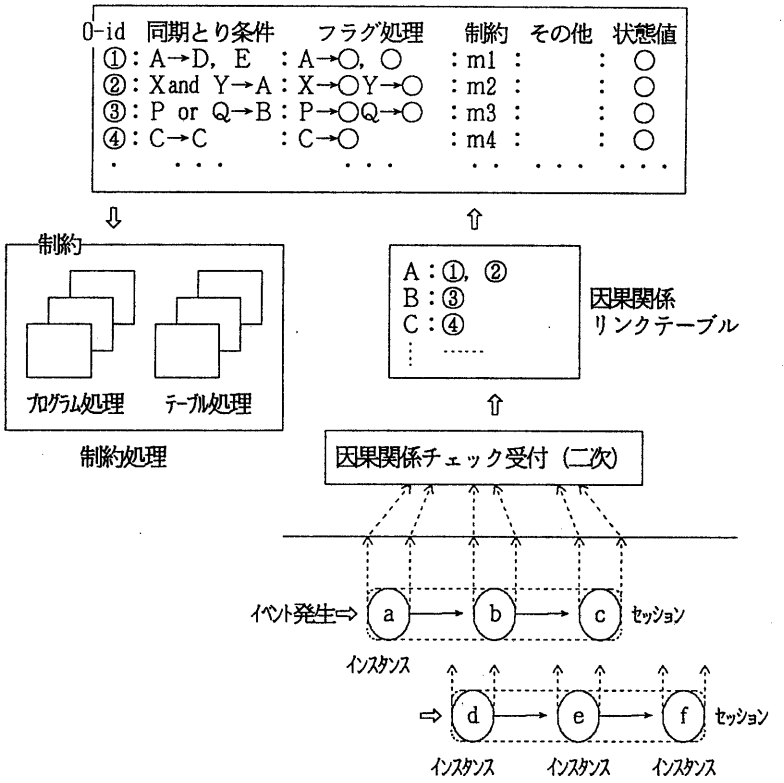
図５の例において２つのセッションにおけるインスタンス（ａ，ｂ，ｃ，ｄ，ｅ，ｆ）は起動時と終了時において毎回外部で定義された因果関係の参照や，状態変更の書き込みを行う。インスタンスｂは起動前に因果関係を参照し，インスタンスｅの終了を確認する。終了していなければ，インスタンスｂはｅの終了まで待ち，その終了確認後，動作するわけである。

そこで，メッセージ受信により起動をかけられたオブジェクト管理（オブ管とも呼ぶ）は，シーケンスデータをみて，オブジェクトに順番に起動をかける。このとき，一つ一つのオブジェクトの起動時，終了時に因果関係チェックをかけるようにする。因果関係チェックは，同期とり条件をみてそのフラグの状態コントロールを行い，必要に応じて制約を起動するようになっている。さらに，異常時には，今動いているオブジェクトの並びを無視して強制的に分岐して①の動作を②に変更しうることが特徴となっている。

### 4.2　因果関係の動機

ＯＣＡにおいて，このような因果関係を導入した理由を述べる。

まず，従来のオブジェクト指向は，一つ一つのオブジェクトがカプセル化により独立しているため，オブジェクト間の因果関係がつけられない。因果関係をつけようとすれば，一つ一つ因果関係間の共通オブジェクトを用意してメッセージパッシングでやりとりしなければならない。その場合，




第５図　　因果関係の制御図（セッションレベル）

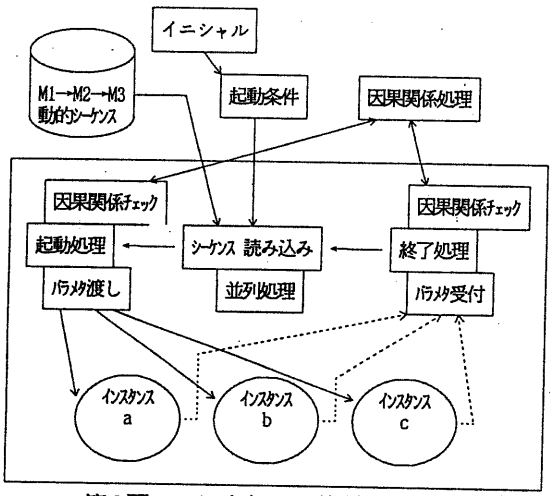
オーバヘッドが大きくなり，性能が犠牲になりやすい。また，因果関係が多いプログラムは共通オブジェクトが増え，プログラムが大きくなる。共通プログラムは，使うオブジェクトを意識して作らなければならない，などの問題点があった。また，従来の手続き型プログラミングの場合，あらかじめイベント間の分岐は用意しておかなければ分岐動作は行われない。別の言い方をすれば特別な状態のときの条件を自由には作れない。つまり複雑な分岐を実行することはできない，という問題点があった。

ＯＣＡでは，因果関係をセッションの外部にて定義することにより上記のような問題点を解決することを意図している。

### 4.3　オブジェクト管理の機能

(1)　オブ管の構成

因果関係を司っているのが図６に示すオブ管処理である。オブ管は，シーケンス読み込み，起動処理，終了処理，および因果関係チェック処理に




第６図　　オブジェクト管理（オブ管）処理

より構成される。シーケンス読み込みにより読み込まれたシーケンスデータに従って，起動条件によりインスタンスオブジェクトを順番に動かす。インスタンスの終了は終了処理でわかる。図内に並列処理と書いてあるのは，オブ管内部の機能でなく，セッション毎にこのオブ管が並列に誕生して，各セッションを取り扱うということである。すなわち，オブ管そのものも実はクラスとして存在しており，セッション毎にオブ管インスタンスが発生するようになっている。

セッションのシーケンスは，動的シーケンスＭ１→Ｍ２→Ｍ３となっているが，各インスタンスＭｉは幾つかのメソッドを組み合わせたクラスからなっているので，インスタンスに起動をかけると，さらにクラスおよびメソッドに対してオブ管が階層毎に発生するようになっている。

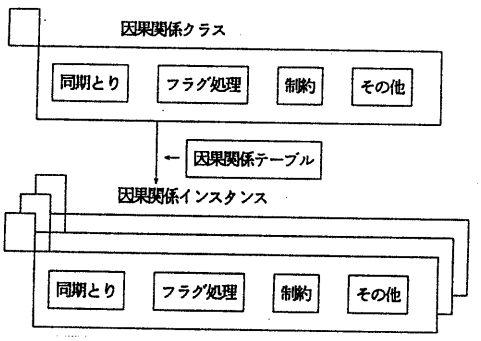
最初にオブ管インスタンスを起動するものがイニシャルにより動かされる“起動条件”である。これには，サイクル起動や時刻指定起動や並列起動などがある。

セッションまたはインスタンスから因果関係のチェックの要請がくると，オブ管は，必要な因果関係のインスタンスを因果関係クラスから誕生させ，一つ一つ因果関係の監視コントロールに入るようにしている。

### 4.4　因果関係タプル

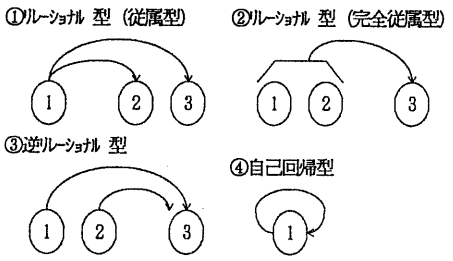
因果関係は，以下に示すような項目からなるタプルに基づいて処理される。（図７）

①同期とり条件は，どのメソッドまたはインスタンスとどのメソッドまたはインスタンスの状態の見方を決めているものである。




第７図　　因果関係のタプル

同期とり条件には，図８に示すようなリレーショナル型（従属型）と（完全従属型），逆リレーショナル型，自己回帰型が考えられる。




第8図　同期とり条件

②フラグ処理とは，同期とり条件に従って，排他制御をやるフラグコントロールをつかさどっている。その条件がそろったときには，必ず制約条件が起動をかけられる。

③その他としては，従属条件による制御条件，優先条件があるときの制約条件，サイクルの条件がくずれたときの制御条件，ロックしたときの制御条件などがおさめられている。

基本的には，同期とり条件や制約条件をオペレータが外部からセットしようとするときに，従属関係や優先順位やサイクル条件やロック条件をチェックしてこのような条件が発生しないようにセットする。しかし，そのチェック条件のどれもをすりぬけたときのにげ道としてその他の条件がセットされる。

## 5. おわりに

本報告においては，新しいモデリング手法として「オブジェクト思考設計法」OCA(Object modeling Consideration Analysis) と呼ばれる手法を述べ，このＯＣＡにおける部品化の考え方，リアルタイム処理を実現するための機構について述べた。特に，リアルタイム処理を行うためには，環境の急激な変化に対応するために，セッションの並列処理と異常時に対する異常処理とが不可欠である。並列に動くセッションおよびインスタンス間で同期をとる機構を各セッション毎に与えるよりも，個々のセッション作成時にはその同期を意識せずに作成し，同期機構を外部から与えるようにしたほうが構成が容易である。この外部から与える同期機構として，因果関係と呼ぶ手法を提案し，その実現法について述べた。

謝辞　本研究の機会を与えてくださった富士通研究所大槻社長に感謝します。

参考文献

〔1〕E.Yourdon, and L.Constantine, Structured Design, Yourdon Press, 1979.

〔2〕S.シュレィアー，S.J.メラー著，本位田，山口訳，オブジェクト指向システム分析，啓学出版，1990.

〔3〕P.Coad, and E.Yourdon, Object-Oriented Analysis, Yourdon Press, 1990.

〔4〕J.ランボー他，羽生田監訳，オブジェクト指向方法論ＯＭＴ，トッパン，1992.

〔5〕龍，佐藤，高原「分散処理における新データモデルの提案」　SITA'90.

〔6〕龍，佐藤，若林「ネットワークシステム構築から見たオブジェクト指向データベースの提案」情報処理学会ｱﾄﾞﾊﾞﾝｽﾄﾃﾞｰﾀﾍﾞｰｽｼﾝﾎﾟｼﾞｭｰﾑ 1991.7.

〔7〕村川，龍「オブジェクトセンサーモデルにおけるメタデータアーキテクチャの提案」情報処理学会ｱﾄﾞﾊﾞﾝｽﾄﾃﾞｰﾀﾍﾞｰｽｼﾝﾎﾟｼﾞｭｰﾑ 1991.7

〔8〕龍，若林，村川「オブジェクトの捉え方とオブジェクトセンサーモデル」情報処理学会ｱﾄﾞﾊﾞﾝｽﾄﾃﾞｰﾀﾍﾞｰｽｼﾝﾎﾟｼﾞｭｰﾑ 1991.7.

〔9〕龍，戸島，村川，豊田「自己組織化通信方式の提案」　SITA'92.

〔10〕豊田，足立，龍「複合オブジェクトを表すセマンティックID方式」情報処理学会データベース研究会 1992.11.

〔11〕市川，龍，戸島「セマンティックIDによるメッセージパッシング方式の提案」情報処理学会ｱﾄﾞﾊﾞﾝｽﾄﾃﾞｰﾀﾍﾞｰｽｼﾝﾎﾟｼﾞｭｰﾑ 1992.12.

〔12〕村川，龍，戸島，豊田「オブジェクト指向設計を支援するハイパー処理機能の提案」情報学ｼﾝﾎﾟｼﾞｭｰﾑ 1993.1.