平成 12 年度

学士学位論文

公理的設計法に基づく RoboCup の設計に関する研究

A Study on Design of RoboCup based on Axiomatic Design

1010432　浜田　真世

指導教員　Ruck Thawonmas 助教授

2001 年 2 月 5 日

高知工科大学 情報システム工学科

要 旨

公理的設計法に基づく RoboCup の設計に関する研究

浜田　真世

本稿は，公理的設計法に基づく RoboCup の設計に関する研究について述べる．従来，ソフトウェア設計に関して，様々な設計法が考えられてきた．しかし，ソフトウェアに限らず様々な「ものをつくる」という作業で行わなければならない設計という作業は，その他科学における事象のように公式化されていなかった．そこで，設計を公式化したものが本稿で扱う公理的設計法である．

公理的設計法は，設計の重要性を考え，工学における機械設計のために MIT の N.P.Suh 教授により提案された設計法である．この公理的設計法に基づくソフトウェア設計の研究も進められている．本稿では，RoboCup クライアントプログラムを対象のソフトウェアとした．RoboCup とは，サッカーを題材に人工知能と知能ロボットの発展を目指す研究分野である．ここで扱うのは、マルチエージェントと Server ／ Client モデルによるサッカーシミュレーションのクライアント部である．また，既存 RoboCup 設計がオブジェクト指向言語である Java 言語を用いていることから，オブジェクト指向技術による公理的設計に基づくソフトウェアシステムについても述べる．

公理的設計に基づき RoboCup の設計を行い，公理的設計の観点から既存プログラムの問題点を検証し，評価を行う．

キーワード　ソフトウェア設計，公理的設計法，RoboCup

Abstract

A Study on Design of RoboCup based on Axiomatic Design

Michiyo Hamada

This thesis is describes design of RoboCup based on Axiomatic Design Theory. Conventionally, various designing methods have been considered for software design. However, not only software but other various "building of a thing " were not well formulated unlike those for describing natural phenomena in Science.

The Axiomatic Design is a design theory that considers the importance of design and was proposed by Professor N.P.Suh of MIT for the machine design in engineering. Research of the software design based on this Axiomatic Design Theory is currently gaining a lot of interest. RoboCup Client is made the target software in this work. RoboCup is a research field that aims at development of artificial intelligence and intelligent robots on the topic of soccer. It is the client of the soccer simulation the multi-agent and the Server/Client model that is treated here. Moreover, since Java, an object-oriented language, is used, design of software systems based on the Axiomatic Design for object-oriented technology is also described.

RoboCup is designed based on an Axiomatic Design and it evaluates by verifying the problem of the existing program from the viewpoint of an Axiomatic Design.

key words Design of Software, Axiomatic Design Theory, RoboCup,

目次

図目次

図目次

第 1 章

序論

設計は，様々なものをを新たに作り出すことを可能とし，ニーズや要望を満たすことが目的である．また，設計とは，様々な工学上の事柄に必要不可欠なものであり，重要である．それは，工学活動の最終結果が設計により決まるからである．工学の初期段階での設計の意志決定は後の全ての結果に影響する．

科学の分野では ”自然の ”プロセスを理解するために，数世紀もの多大な努力の結果，自然法則と原理を「公式化」するという歴史上重要な発見が多くあった．これに対し，様々な「ものをつくる」という「創造的プロセス」に必要かつ重要である設計という作業は，科学的基礎の上に成り立たないと仮定されていた．そして，設計には系統的かつ科学的基礎の上で扱う ”原理 ”と ”法則 ”が公式化されておらず，また，共通の判断基準がないために，設計活動の成果を合理的に理解できなかった．ここで，良い設計解と容認できない設計解が存在するという事実から，それらを区別するために良い設計解の特徴や属性などの共通要素により，プロセスを統合する理論として考えられた方法が,「公理的設計法」である．

近年情報化が進み，様々なソフトウェアが大量に生産され利用されるようになった．このソフトウェアやソフトウェアシステムの設計も同様に重要である．ここで，本稿では，当初機械設計のためにマサチューセッツ工科大学の Num.P.Suh 教授により考えられた「公理的設計法」を用いたソフトウェア設計を行う．設計を行うソフトウェアは，RoboCup というサッカーを題材に進められる研究分野における，サッカーシミュレーションのクライアントプログラムである．このプログラムには様々な問題がある．公理的設計法を用いた設計により，公理的設計法の利点を活かし，既存プログラムの問題点を解決することを目的とする．また，設計結果と既存プログラムの比較を行うことで，問題の検証を行う．

まず，第２章では，公理的設計法の背景，及び，公理的設計法について示す．２つの設計公理をはじめとする公理的設計法における特有の手法及びそれらを用いた設計の手順を要所で例を挙げ述べる．

第３章では，ソフトウェアの公理的設計について述べる．公理的設計法に基づくソフトウェア設計での設計プロセス及び，そのプロセスにおける設計行列，設計方程式，Module-Junction 図，フローチャートの役割，特性，作成法などを述べる．また，設計を行う RoboCup プログラムがオブジェクト指向方法論において用いられる Java 言語を用いて実装を行うことから，公理的設計法を用いたソフトウェア設計の中でも公理的設計法を用いたオブジェクト指向のソフトウェア設計について論じる．また，それらの作成法は，例を挙げ手順を明確にする．

第４章では，公理的設計法を用いたソフトウェア設計として，既存の RoboCup クライアントプログラムの再設計を行う．初めに、その後，RoboCup 概要，及び，公理的設計法を用いたオブジェクト指向による RoboCup の設計を行う．

第５章では，既存プログラムの問題点を公理的設計法の観点から検証する．

第 2 章

公理的設計法概要

公理的設計法には,「設計公理」や「設計行列」をはじめとする多くの設計概念がある．この章では，設計作業の重要性から考えられた公理的設計法の概念，特徴，設計プロセスの概略 [1] を示す.

2.1　公理的設計法とは

公理的設計法とは，設計の重要性から設計作業という創造的プロセスを公式化したものである．設計作業の公式化とは，良い解と容認できない設計解が存在するという事実をもとに，それらを区別するために良い設計解の特徴や属性などの共通要素により，プロセスを統合することである．このように設計作業を公式化し，統一化することの利点は，第１に，容易に最良の設計解が得られるという点である．次に，無駄がなく，容認できない設計より維持，変更，拡張が容易であることが挙げられる．そして，技術の進歩と共に増大なものとなる顧客の要求に対応するための品質の向上，機能の追加，管理などの様々な変化に的確に対応することが可能である点である.

2.2　公理的設計法における設計世界

公理的設計法では，設計世界を４つの領域に分類している．それらの領域はそれぞれの特性があり，領域ごとに各変数を持つ．また，領域間には密接な関わりがある，

2.2.1　設計世界の４つの領域

公理的設計法において，設計世界には４つの領域がある.（図 2.1）

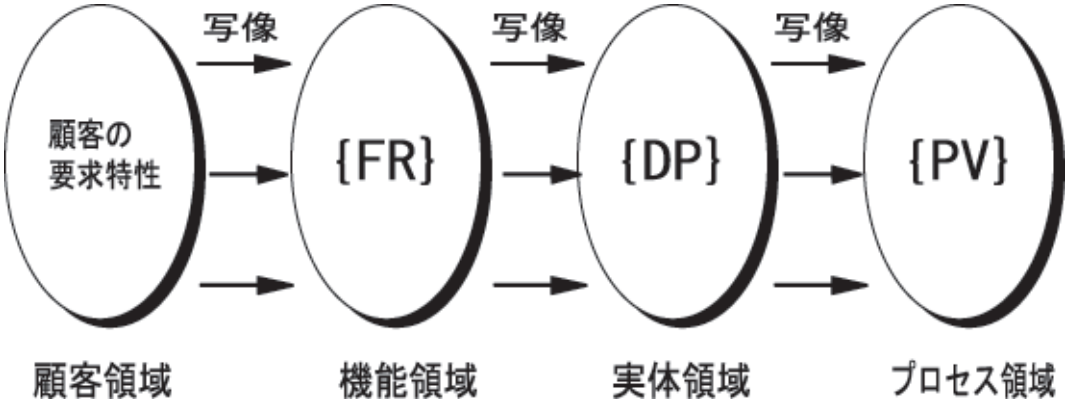

図 2.1　設計世界の４つの領域

２つの隣接する領域間の関係（図 2.2）は，ある領域を「何を達成したいのか（What）」見ると，その右隣の領域は「それをどうやって実現するか（How）」を示すという相対的な関係がある.

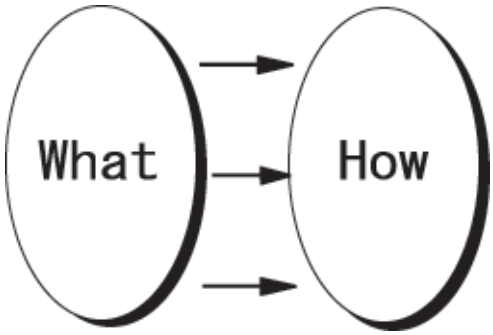

図 2.2　相互領域間の関係

図 2.1，図 2.2 におけるこの関係は，設計の目標が常に機能領域で設定され，一方具体的な解は常に実体領域で作られるということを示している．公理的設計における設計手順には，設計プロセスの各階層レベルで２つの領域を結び付けることが含まれる．また，この２つの領域は，本質的に互いに独立である.

図 2.1 の各領域内における各変数について以下に示す.

- CA(Customer Attribute) : 顧客の要求特性（以下 CA）
- FR(Functional Requirement) : 必要機能（以下 FR）

  設計目標の特徴を完全に表すような最小の独立な必要条件
- DP(Design Parameter) : 設計変数（以下 DP）
- PV(Process Variable) : プロセス変数（以下 PV）
- C(Constraint) : 制約条件 C

  ある値以下であればよいという制限値．他の C や FR と独立である必要がない．

2.3　設計公理

公理的設計法では，FR と DP 間の写像によって解の創造を行うが，FR を満たす DP の生成から複数の設計解が生まれ得る．この無限にある設計解の中から最良の設計解を導くために，２つの公理と７つの系，16 の定理（付録 A 参照）が与えられている．

複数の設計解の中から最良の設計を決定するための設計公理は２つある．

公理１ ：独立公理

必要機能 FR の独立性を保て

公理２ ：情報公理

情報量を最小化せよ

2.3.1　独立公理

公理１の独立公理は，機能と物理的変数の関係を決めるものであり,「必要機能 FR の独立性を保て」である．これは，設計プロセスにおいて，機能領域における FR を満たすように実体領域における DP を決めていく時，DP が対応する FR のみを満たすような写像関係にしなければならないというものである．

2.3.2　情報公理

公理２の情報公理は，設計の複雑さについてであり,「情報量を最小化せよ」である．これは，独立公理を満たす全ての設計のうち，情報量が最小のものが最良の設計であるというものである．

情報公理によれば，最小の情報内容の設計は，最も良い設計である．ソフトウェア設計における情報量は，複雑さと等しいと見なされ，指定された FR を達成することの確率に関して定義される．これによれば，FR を達成する確率の低い設計は，高い確率のものより複合的で更に高い情報内容を持つこととなる．そして，長いソフトウェアプログラムを持つソフトウェアは短いものよりコードの増大によりミスを犯すことの確率が高いため，高い情報内容を持つというこがいえる．また，ソフトウェアにおいて,「バグ」を持つということは FR が十分にないことを意味するため，複数の設計における情報内容の判断に用いることができる．

2.4　設計の分類

公理的設計法では，設計を３つに分類している．独立設計（uncoupled design），準独立設計（decoupled design），干渉設計（coupled design）の３種である．独立設計は，独立公理を満たしている．一方干渉設計はある機能が他の機能に依存していることであり，この意味で独立公理に反している．そして，干渉設計は準独立設計にすることが可能である．準独立設計は，必要な情報量が最小ではないので，独立設計には劣る．

2.5　設計行列と設計方程式

公理的設計法では，独立公理の数学的表現として，FR と DP をベクトルで表し，これらの間の関係を設計行列（Design Matrix）という正則行列で表す．また，FR，DP の関係を設計行列で示したものを設計方程式と呼ぶ．以下に設計行列と設計３種を示す例を挙げる．以下の例で X はゼロでない値を表し，ある DP を変更すると対応する FR に影響が及ぶこ

とを示す.

$$\begin{Bmatrix} FR1 \\ FR2 \\ FR3 \end{Bmatrix} = \begin{bmatrix} X & 0 & 0 \\ 0 & X & 0 \\ 0 & 0 & X \end{bmatrix} \begin{Bmatrix} DP1 \\ DP2 \\ DP3 \end{Bmatrix} \quad \text{独立設計（}\mathbf{UncoupledDesign}\text{）}(2.1)$$

$$\begin{Bmatrix} FR1 \\ FR2 \\ FR3 \end{Bmatrix} = \begin{bmatrix} X & X & 0 \\ X & X & X \\ 0 & X & X \end{bmatrix} \begin{Bmatrix} DP1 \\ DP2 \\ DP3 \end{Bmatrix} \quad \text{干渉設計（}\mathbf{CoupledDesign}\text{）}(2.2)$$

$$\begin{Bmatrix} FR1 \\ FR2 \\ FR3 \end{Bmatrix} = \begin{bmatrix} X & 0 & 0 \\ X & X & 0 \\ X & X & X \end{bmatrix} \begin{Bmatrix} DP1 \\ DP2 \\ DP3 \end{Bmatrix} \quad \text{準独立設計（}\mathbf{DecoupledDesign}\text{）}(2.3)$$

独立設計（式 2.1）の示す設計行列は，対角要素のみの対角行列である．干渉設計（式 2.2）が示す設計行列は，対角要素の他に上三角部分に非対角要素を持つ行列である．準独立設計（式 2.3）が示す設計行列は，設計行列が対角要素の他に下三角部分に要素を持つ行列である．

ここで，独立設計と干渉設計の例として，日常よく使用する洗面などの水栓器具を挙げる．図 2.3 の左図ような 2 ハンドル式の水栓器具は，左右のハンドル両方を回すことにより，

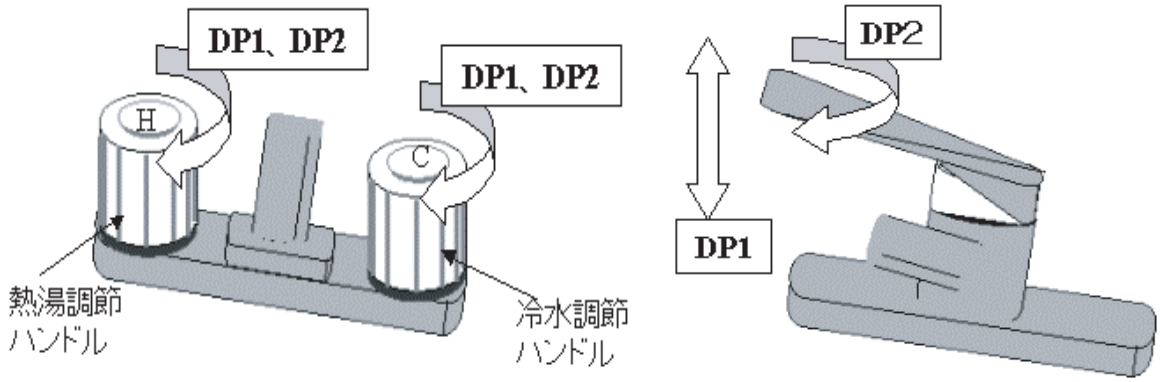

図 2.3　例：２つの水道器具

流水量と水温を調節する仕様のものである．一方，図 2.3 の右図ような水栓器具はシングル

レバー混合栓と呼ばれ，中央レバーの上下により流水量の調節を行い，そして左右の加減で水温の調節を行う仕様のものである．この 2 つの水栓器具において，満たすべき必要機能 FR は，式 2.4 である．

$$\begin{cases} FR1 = \text{吐水量の調節} \\ FR2 = \text{温度の調節} \end{cases} \tag{2.4}$$

DP は FR を実現するために必要な動作でなければならない．それぞれの設計に対する，DP は右図の場合が式（2.5）であり，左図の場合が式（2.6）である．

$$\begin{cases} DP1 = \text{左右のハンドル両方を回す} \\ DP2 = \text{左右のハンドル両方を回す} \end{cases} \tag{2.5}$$

$$\begin{cases} DP1 = \text{レバーの上下により調節} \\ DP2 = \text{レバーを左右に回す} \end{cases} \tag{2.6}$$

これらそれぞれについて，設計方程式を作成すると，式（2.4），式（2.5）における設計行列は，式（2.7）となり「干渉設計」となっている．これは容認できない設計解である．

$$\begin{Bmatrix} FR1 \\ FR2 \end{Bmatrix} = \begin{bmatrix} X & X \\ X & X \end{bmatrix} \begin{Bmatrix} DP1 \\ DP2 \end{Bmatrix} \quad \text{干渉設計（}\mathbf{CoupledDesign}\text{）} \tag{2.7}$$

それに対し，式（2.4），式（2.6）の設計行列は，式（2.8）となり「独立設計」である．これは良い設計解といえる．

$$\begin{Bmatrix} FR1 \\ FR2 \end{Bmatrix} = \begin{bmatrix} X & 0 \\ 0 & X \end{bmatrix} \begin{Bmatrix} DP1 \\ DP2 \end{Bmatrix} \quad \text{独立設計（}\mathbf{UncoupledDesign}\text{）} \tag{2.8}$$

このように，公理的設計法では設計行列を用いることで容易に機能の独立の確認を行うことが可能である．

2.6　設計手順と独立公理

独立公理は，独立設計を早く実現できるという点で，設計手順に関しても深い意味を持つ．既存の設計を分析する場合には，独立公理に反してないかどうか設計行列を作り確認する必要がある．また，まったく新しい設計を行う場合でも，構想段階で独立公理を有効に使うことが出来る．

2.6　設計手順と独立公理

設計プロセス全体を通じて，常に，設計公理や系，定理を心に留め，適用できる場合は常に適用しなければならない．対角行列や三角行列の設計行列となる DP を求めることの必要性は，いくら強調してもしすぎることはない．

第 3 章

ソフトウェアの公理的設計

この章では，ソフトウェアの公理的設計について述べる．ソフトウェア設計では，２章において述べたような一般的公理的設計と異なり，設計プロセスに Module-Junction 図やフロー図の作成が加わる．ソフトウェア設計の公理的設計プロセスの役割，特性，作成法などを述べる．また，設計を行う RoboCup がオブジェクト指向言語である Java 言語により実装されることから，オブジェクト指向技術と公理的設計法の結合であるオブジェクト指向ソフトウェアシステム（Axiomatic Design of Object- Oriented Software Sostem:ADo-oSS）[2] の公理的設計についても述べる．

3.1　ソフトウェア設計の重要性

ソフトウェアは，近年，我々の生活において至るところに存在し，様々なものを形成，制御し重要な機能を果たしている．これらソフトウェアが崩壊する時，その問題が解決するまで世界はほとんど静止に至る．生活上重要な役割を果たすソフトウェアの設計に対しても，設計理論の使用を通して，ソフトウェア設計作業を正確に，完全に行うことが重要である．公理的設計は，複雑なソフトウェアシステムのための特に強力なツールである．

3.2　ソフトウェア開発手順

一般的なソフトウェア開発の手順を以下に示す．

1. 必要性の分析，及び，要求，機能，データ，ツールなど様々な情報の収集

2. 設計：ソフトウェアの抽象的な設計
3. コーディング：設計に従ってプログラムを作成する
4. デバッグ：エラーを訂正する
5. テスト：サイト実験
6. メンテナンス：リリース後の保守
7. 拡張：ソフトウェアに新たな機能，特徴を追加するための開発の主要な拡張

これまでに述べたように，ソフトウェア開発では，要求を正確に満たし無駄のないソフトウェアの作成，そして、将来の機能の拡張，変更，訂正のためにも手順における（１）と（２）が最も重要である．

3.3　ソフトウェアのための公理的設計プロセス

ソフトウェア開発において最も重要とされる設計プロセスに公理的設計法は有効である．以下に公理的設計法に基づくソフトウェア設計手順を示す．

1. ソフトウェアシステムの CA の決定と FR の定義
2. 領域間のマッピング
3. 設計公理と設計行列による選択
4. 最下レベル（leaf レベル）までの分解
5. Module-Junction 図の作成
6. フロー図の作成

これより，それぞれの設計プロセスの詳細と２つの公理との関わりを述べる．

3.3.1　（１）ソフトウェアシステムの FR を定義する

ソフトウェアシステム設計をする際の最初のステップは，ソフトウェアシステムが満足させなければならない顧客領域における CA の決定である．そして次に、CA を満たすように

ソフトウェアの FR，及び，C を決定する．FR は，前章で述べた公理的設計法によるの設計と同様にソフトウェア設計においても，設計目標の特徴を完全に表すような最小の独立な必要条件でなくてはならない．そして，DP については考慮せず忠実に顧客要求を満たすように決定されなければならない．

3.3.2　（２）領域間のマッピング

公理的設計に基づくソフトウェア設計の次のステップは，DP を確認することによって，機能領域の FR を物理領域にマッピング（写像）することである．また，DP は C と一致しているもの程選択されなければならない．DP が選択されると，次にプロセス領域へ移行し，PV の確認を行う．PV はサブルーチン，または，機械コード，コンパイラによって確認される．このマッピングプロセス間，常に独立公理を満足しなければならない．

3.3.3　（３）設計行列と設計方程式による確認，選択

各階層の FR，DP の分解とマッピング作業において，それぞれの階層で設計行列を作成し，独立設計または，準独立設計を保つように FR と DP を選択しなければならない．ここで，重要である設計公理は独立公理である．

3.3.4　（４）最下位レベル（leaf レベル）までの分解

FR，DP は，リーフレベルに分解されなければならない．そこで，その設計は，更なる分解なしで実行され得る.（参照：図 3.1）

Leaf とは，FR 分解における最終レベルの FR である．Leaf レベルとは，分解プロセスの FR 木構造において，全ての枝が Leaf である状態である．この状態に達した時，設計プロセスは終了する．

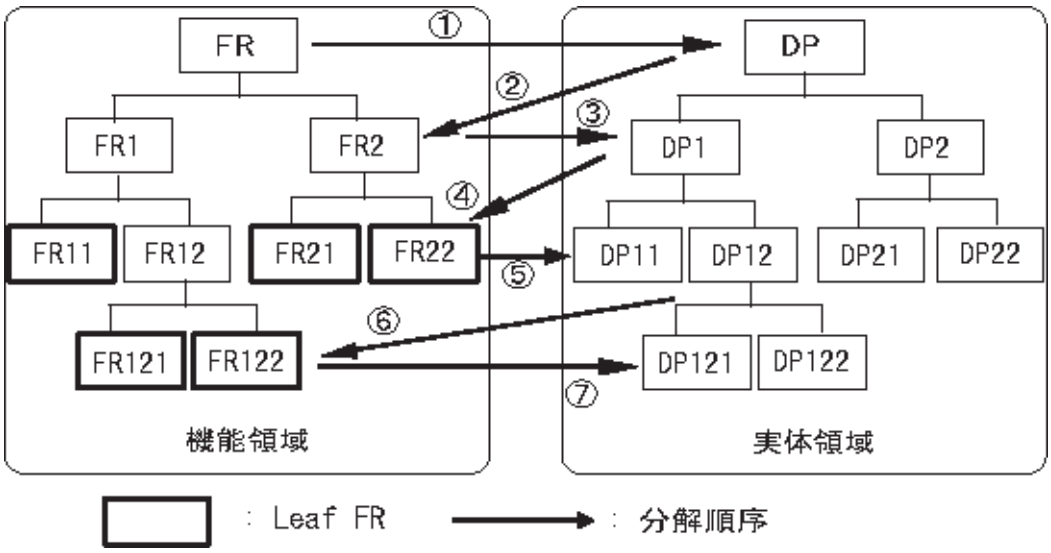

図 3.1　リーフの状態

3.3.5　（５）Module-Junction 図の作成

Module-Junction 図は，モジュール間の関係を表す方法として提示される．この Module-Junction 図の作成は，前段階で作成した設計行列を元に行われる．また，分解最終段階の木構造における Leaf のモジュールのみからなる．独立設計を表すためには S 結合が用いられ，準独立設計は C 結合，干渉設計は F 結合で表現される．

3.3.6　（６）フローチャート表現

フローチャート（フロー図）はソフトウェア全体のシステムアーキテクチャを表現する方法として提示される．また，ソフトウェアの最終設計は，フローチャートによって表され，ソフトウェア開発フェーズ間の管理手段として使うことができる．このフロー図は，分解最終段階の木構造における leaf のモジュールのみからなり，前手順での，Module-Junction 図を参考に作成される．このフロー図に基づき，個々のモジュールがソフトウェアシステムを作成するために，開発され，連結される．また，それらの動作を繰り返すことにより，最高位の FR を産出することが出来る．

3.3.7　フローチャートのアプリケーション

フローチャートで表現されたソフトウェアのためのシステムアーキテクチャは，多くの異なる方法に使用することができる．以下に使用例を挙げる．

1. ソフトウェア故障の診断
2. ソフトウェアの変更命令

   １つのある変更が提案された場合，変更されなければならない全ての関連モジュールを確認するために使用することができる．
3. ソフトウェア開発における作業の割当と管理

   開発計画を組織化し使用することができる．プロジェクト実行と管理を援助する．

公理的設計法に基づくソフトウェア設計は，変更，修正，拡張が容易である．これは，マッピング及び，分解の各ステージで正しい決定を行う結果である．

3.4　オブジェクト指向ソフトウェアシステムの公理的設計

オブジェクト指向ソフトウェアシステムの公理的設計（Axiomatic Design of Object-Oriented Software System： ADo-oSS）[2] は，機械設計のために考えられた公理的設計法をオブジェクト指向のソフトウェアシステムに適用するために進められている研究である．この ADo-oSS は，オブジェクト指向技術（Object-Oriented Technique：OOT，以下 OOT）と公理的設計を結合することで，ソフトウェア開発を科学の主題にし，デバッグと広い変更の必要性を減少させる，もしくは消去することを目的としている．

公理的設計法の観点では，良いソフトウェア設計とは，モジュール関係が独立で，明白に動作するように設計されているものである．従来，ソフトウェアの分解はプログラマの経験に基づき，またそれが頭の中において行われているため，必要機能 FR，マッピング（領域間の写像）や往復による分解と階層構造，及び設計行列を明白に定義しない．これらの従来の欠点は，公理的設計法に基づくソフトウェア設計行うことによって克服することが可能で

ある．公理的設計法は，モジュールが正しく，正しい要求で正しい場所に独立で定義され，決定されるということを保証する．

3.5　ソフトウェアの V モデル

図 3.2 の「ソフトウェアの V モデル」は，オブジェクト指向ソフトウェアシステムの公理的設計の概念を概略的に示す．この図において，前半の段階は，ソフトウェアを設計することを示している．公理的設計の top-down アプローチによりソフトウェア階層を構築する．その後，モジュールを定義するために設計行列を生成する．最終段階は，設計されたシステムのために公理的設計におけるフローチャートを作成する bottom-up アプローチによりオブジェクト指向のモデルの作成である．これらの手順により公理的設計法に基づくオブジェクト指向ソフトウェアシステムが作成される．

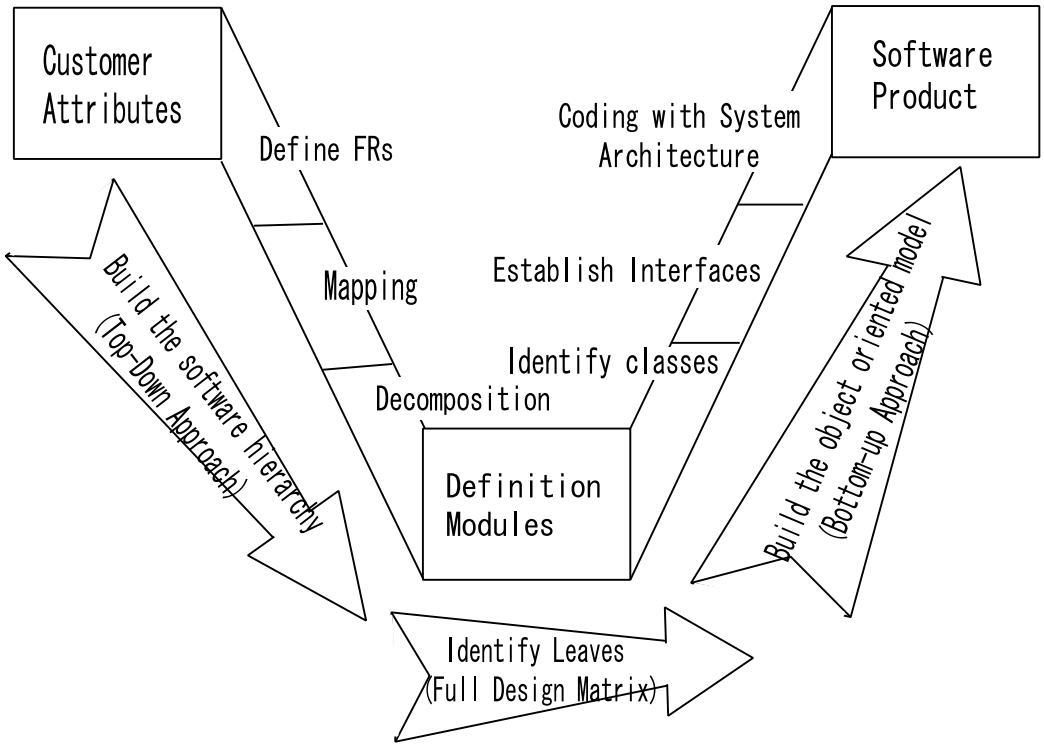

図 3.2　ソフトウェアの V モデル

3.6　オブジェクト指向技術と公理的設計法

OOT は系統的な方法において，公理的設計法における領域間での往復による設計，及び，分解など階層型性質を扱うことができない．従って，OOT には，それに使われている様々なオブジェクトやクラス，スーパークラスなどのキーワードと，公理的設計に使われるものと同等の関係を示す基準が必要となる．

OOT における「スーパークラス」,「クラス」,「オブジェクト」などの概念において，これらの間の区別はシステム設計の連続する層で FR を扱うために必要である．公理的設計と OOT の用語でのそれぞれ相互に相当するものとして，以下のような関係が定義されている．ここで，オブジェクト指向における「オブジェクト」と公理的設計法の関連付けを述べる．

オブジェクト指向方法論での基礎的構成は「オブジェクト」であり，これは公理的設計法での FR に相当する．１つの構成要素におけるオブジェクトは，データ（DP に相当する）とメソッド（FRi と DPi 間の関係，すなわちモジュールに相当する）により構成される．また，オブジェクトは，オブジェクト内にあるデータ及び，メソッドにより供給された入力を用い，いかに正確に動作を行うかに関する明確な情報を保持する．公理的設計法において，これは，$FR_i = A_{ji} DP_j$ が相当する．図 3.3 において，オブジェクトを図式化したものを示す．

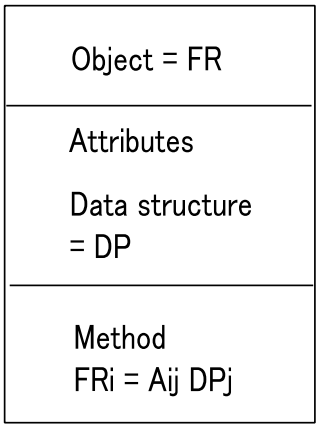

図 3.3　オブジェクト図

3.7　オブジェクト指向ソフトウェアシステムの公理的設計の基本

図 3.2 において示されたオブジェクト指向ソフトウェアシステムの公理的設計（Ado-oSS）の設計手順は，次のステップを包括する．

1. ソフトウェアシステムの CA の決定，及び，CA を正確に満たす機能領域におけるソフトウェアの FR，C の定義
2. DP を確認することにより，物理領域に機能領域の FR をマッピング，及び，設計行列・独立公理による設計の評価
3. Leaf レベルまでの FR，DP の領域間の往復による分解
4. モジュールの定義
5. オブジェクト，属性，及び，オペレーションの確認．

以上の設計手順において,（１）から（３）まではこれまでに示した示した公理的設計法によるソフトウェア設計の手順と同等である．以下においてステップ（４）及び（５）について補足する．

3.7.1　（４）モジュールの定義

公理的設計枠組の最も重要な特徴の１つは設計行列である．設計行列は，FR，DP 間の関係を提供するからである．ソフトウェア設計の場合，設計行列はソフトウェアを作成する際の２つの重要な基礎を提供する．１つの重要な基礎は，設計行列における各要素がオブジェクト指向方法論に関するメソッド（あるいは，オペレーション）を示すことである．もう１つの重要な基礎は，ある DP が与えられた場合，特定の FR を満足させるための設計行列における各行がモジュールを表すことである．大部分のケースにおいて，多くの干渉が対角でない要素から起こる．しかし，設計行列が三角行列であるならば，適切な変更のシーケンスが干渉設計を回避することが可能である（準独立設計）．

3.7.2　（５）オブジェクト、属性、及びオペレーションの確認

設計階層における全ての DP が FR を満足させるために選択されるため，オブジェクトを確認することは比較的容易である．Leaf はある分解階層木において最も低いレベルのオブジェクトである．しかし，異なる分解木に属するリーフレベルオブジェクトは，同じ階層レベルであるとは限らない．オブジェクトの定義として DP，オペレーション，モジュールがオブジェクトのために，そしてオブジェクトモデルを組み立てるために定義されなければならない．

3.7.3　線画プログラムによる例

以上の設計手順の実行例として,「シンプルな線画プログラムのソフトウェア設計」挙げる．この線画プログラムの要求は,「直線を引くことができるソフトウェアプログラムの設計．また，その結果生じる公理的設計をオブジェクト指向の方法に変換する」ことである．

まず，最高位の FR は以下のように定義することができる．

$$\begin{cases} FR1 : \text{線画要素（ラインエレメント）の定義} \\ FR2 : \text{描画位置の検出} \\ FR3 : \text{ラインエレメントをウィンドウに通す} \end{cases} \tag{3.1}$$

これら FR に一致する DP は以下のように選択できる．

$$\begin{cases} DP1 : \text{ライン特性} \\ DP2 : \text{マウスクリック情報} \\ DP3 : \text{図のための } GUI \end{cases} \tag{3.2}$$

最高位の FR（3.1）および DP（3.2）の設計行列は，

$$\begin{Bmatrix} FR1 \\ FR2 \\ FR3 \end{Bmatrix} = \begin{bmatrix} A_{11} & 0 & 0 \\ 0 & A_{22} & 0 \\ A_{31} & A_{32} & A_{33} \end{bmatrix} \begin{Bmatrix} DP1 \\ DP2 \\ DP3 \end{Bmatrix} \tag{3.3}$$

次に，最高位の FR 及び DP は，以下のように分解される．

第 2 レベルの FR1:

$$\begin{cases} FR11 : \text{スタートの定義} \\ FR12 : \text{エンドの定義} \end{cases} \tag{3.4}$$

第 2 レベルの DP1:

$$\begin{cases} DP11 : \text{スタートポイント} \\ DP12 : \text{エンドポイント} \end{cases} \tag{3.5}$$

第 2 レベルの FR1（3.4）および DP1（3.5）の設計行列は,

$$\begin{Bmatrix} FR11 \\ FR12 \end{Bmatrix} = \begin{bmatrix} a & 0 \\ 0 & b \end{bmatrix} \begin{Bmatrix} DP11 \\ DP12 \end{Bmatrix} \tag{3.6}$$

第 2 レベルの FR2:

$$\begin{cases} FR21 : \text{マウスプッシュの検出} \\ FR22 : \text{マウスリリースの検出} \end{cases} \tag{3.7}$$

第 2 レベルの DP2:

$$\begin{cases} DP21 : \text{プッシュのためのイベント} \\ DP22 : \text{リリースのためのイベント} \end{cases} \tag{3.8}$$

第 2 レベルの FR2（3.7）および DP2（3.8）の設計行列は,

$$\begin{Bmatrix} FR21 \\ FR22 \end{Bmatrix} = \begin{bmatrix} c & 0 \\ 0 & d \end{bmatrix} \begin{Bmatrix} DP21 \\ DP22 \end{Bmatrix} \tag{3.9}$$

第 2 レベルの FR3:

$$\begin{cases} FR31 : \text{図環境の準備をする} \\ FR32 : \text{ラインを引く} \end{cases} \tag{3.10}$$

第 2 レベルの DP3:

$$\begin{cases} DP31 : \text{ウィンドウタイプ} \\ DP32 : \text{グラフィックス情報} \end{cases} \tag{3.11}$$

第 2 レベルの FR3（3.10）および DP3（3.11）の設計行列は,

$$\begin{Bmatrix} FR31 \\ FR32 \end{Bmatrix} = \begin{bmatrix} e & 0 \\ f & g \end{bmatrix} \begin{Bmatrix} DP31 \\ DP32 \end{Bmatrix} \tag{3.12}$$

以上の分解により，以下図（3.4）のように OOT フォーマットによりオブジェクト構成を得ることができる.

図 3.4 において，４段で構成されるオブジェクトボックスにおいて，上段は FR のためのオブジェクト名である．第２段のボックスは，第２レベル FR，すなわち，FR11，FR12，

	Object 1 (for FR1)		Object 2 (for FR2)		Object 3 (for FR3)	
Second Level Objects (Behavior in OOT)	Object 11 (for FR11) : Define start	Object 12 (for FR12) : Define end	Object 21 (for FR21) : Detect mouse push	Object 22 (for FR22) : Detect mouse release	Object 31 (for FR31) : Prepare the drawing environment	Object 32 (for FR32) : Draw the line
Attribute or Data	DP11 : Start point	DP12 : End point	DP21 : Event for push	DP22 : Event for release	DP31 : Window type	DP32 : Graphics Information
Second Level Objects (Behavior in OOT)	a*DP11	b*DP12	c*DP21	d*DP22	e*DP31	f*DP32

図 3.4　オブジェクトの２つの異なるレベル（オブジェクト構成）

FR21 等のためのものである．第３段のボックスは，”data”（DP11，及び，DP12 など）のためのものである．第４段のボックスは，メソッド（またはオペレーション）すなわち，a*DP11，b*DP12 などのためのものである．また，将来の再利用のために親のレベルの関係を保つことが可能である．これは，(FR1-A11-DP1)，(FR2-A22-DP2)，(FR3-A33-DP3) であり，これらは，各関係が基本的なオブジェクト構造であることを明白にするため，再利用可能を保証している．ここで，(FR1-A11-DP1)，(FR2-A22-DP2)，(FR3-A33-DP3) は，式 3.3 における，対角要素と FR，DP との関係を示している．

式 3.4 において,（FR1-A11-DP1）は FR1 のための Object 1d,（FR2-A22-DP2）は FR2 のための Object 2d,（FR3-A33-DP3）は FR3 のための Object 3d，として指定される．”d”は，それらが対角要素のみを表すということを表示するために，オブジェクト番号に加えられる．これに対し，非対角要素を表示するには，アスタリスクがオブジェクト番号に加えられる．例えば，$FR3 = M3 \cdot DP3$ の持つ非対角要素（FR3-A31-DP1）(FR3-A32-DP2) の表示は，FR3*のために Object 3*のように示される．

また，この関係は，OOT 表記法では，以下図 3.5 のように表される．

FR 及び DP を持つ十分な設計行列は，OOT 構造に変換することが可能である．図 3.6

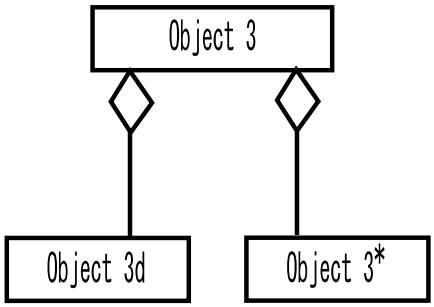

図 3.5　Object Diagram

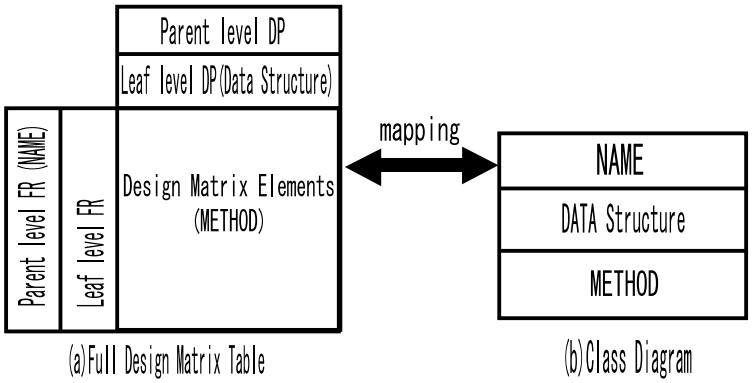

図 3.6　OOT への変換

また，２つの同レベルの FR を基準とし，下位２つのレベルまでの設計行列の構造を示す図により，容易にオブジェクトに分ける事ができる．上記方法と同じであるが，設計行列における下位レベルとの相互関係も示し確認することができる．図 3.7 により例を示す．上図では，まず対角要素のみからなる部分が独立したオブジェクトとなる（Ed）．そして，その次にそれ以下の対角要素と非対角要素のオブジェクトの集合という形になる（Fd, F*）．この分類を上位レベルまで進めることで，完全な設計行列を用いたオブジェクト図が完成する．

以上のオブジェクトの相互関係が示されると，次はオブジェクト図を元にクラス図の作成を行う．この場合，上記図 3.5 のように非対角要素と対角要素のみのオブジェクトの集合の結合により作成する．

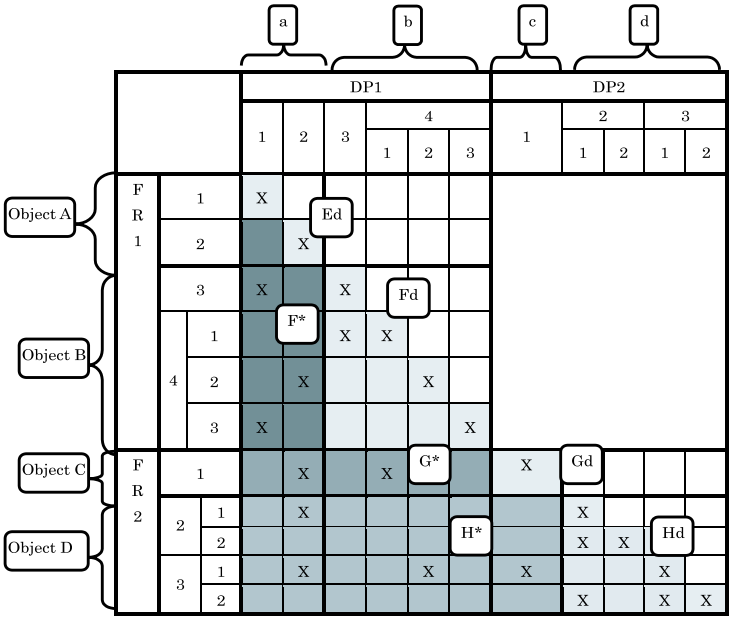

図 3.7　Full Design Matrix

第 4 章

公理的設計法による RoboCup の設計

この章では，3 章において示したソフトウェアの公理手設計法を用い，既存の RoboCup クライアントプログラムの再設計を行う．まず，RoboCup 概要を示し，その後，公理的設計法による RoboCup の設計結果を示す．

4.1 RoboCup とは？

RoboCup は，人工知能と知能ロボットに関する研究を促進，融合，発展するためにサッカーを題材として，日本の研究者らによって提唱された研究分野である．[5] ロボットやソフトウェアエージェントにより構成されたチームによってサッカーが実際に実行されるためには，自律エージェント，マルチエージェントによる，協調，戦略の獲得，実時間処理，そしてセンサー技術などといった多様な技術を組み入れる必要があり，RoboCup では，これらの課題をソフトウェアロボットによるシミュレーションと現実世界で動作するロボットの開発を通じて解決していくことを目的としている．また，現在では，サッカーだけでなく，大規模災害へのロボットの応用としてレスキュー，次世代の技術の担い手を育てるジュニアなどが組織されおり，将来は，2050 年には FIFA のチャンピオンにヒューマンノイドリーグのチームが勝つことが目標とされている．

4.1.1　RoboCup サッカー

RoboCup で一番初めに組織されたものが RoboCup サッカーであり，2000 年現在自律移動ロボットの３リーグ，シミュレーションの１リーグの全４リーグある．本研究対象は，シミュレーションリーグである．このシミュレーションリーグは，RoboCup サッカーの中では一番古くから存在するリーグで，コンピュータ上に設営した仮想フィールドでソフトウェアのプレイヤーがサッカーをするものである．

4.1.2　RoboCup サッカーのモデル

試合は，通産省電子技術総合研究所で開発されたサッカーサーバを利用し，Server/Client 方式で行われる．サーバ（サッカーサーバ）は仮想的なフィールドを提供し，ボールとプレイヤーの全ての動きをシミュレートする．各クライアントはそれぞれのプレイヤーの動きをコントロールし，プレイヤーの頭脳の役割をする．サーバと各クライアントとの通信は UDP/IP によるソケットを通じて行われる．

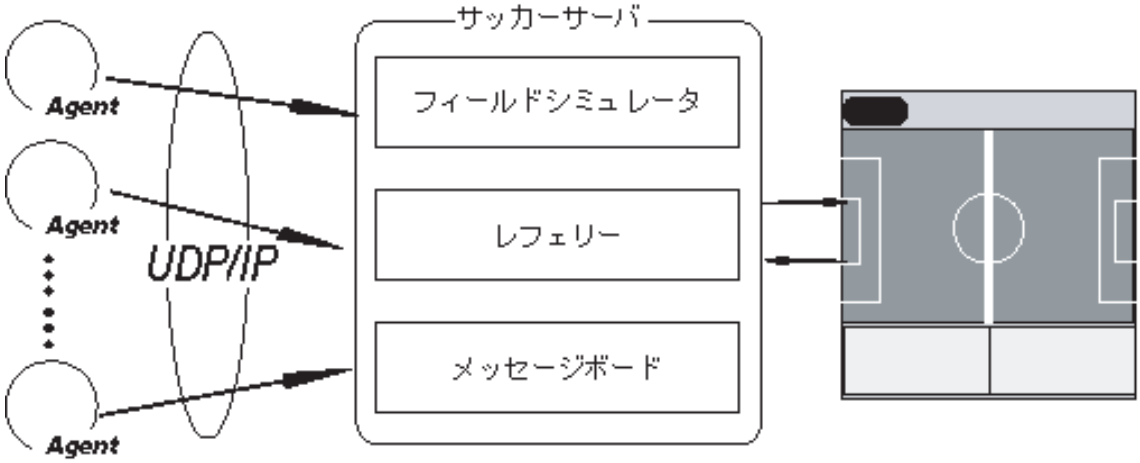

図 4.1　サッカーサーバーのモデル

サッカーサーバはサッカーフィールド，ボール，審判などプレイヤー以外の全てをシミュレートする,「メッセエージボード」,「フィールドシミュレータ」,「審判」の３つにより構成されている．また，これらの情報を画面で見て確認できるようにするためにサッカーモニタ

（soccer monitor）に情報を送り画面表示させる．

RoboCup では，複数のエージェントが協調して，単独で解決できない問題を解決するマルチエージェントシステムが研究され採用されている．１つのクライアントは原則として，１つのエージェントだけを制御しなければならない．つまり１つのクライアントが複数のエージェントの情報を集中制御するようなことは許可されておらず，１人のプレイヤーを１クライアントとして作成する．

4.1.3　サッカーサーバとの通信

クライアントとサッカーサーバとの通信内容は以下ような構成である．

クライアントからサーバへの通信内容

- サーバへの接続以来：コマンドにより「送信」
- 自分の選択した「行動」

サーバからクライアントへ通信内容

- サーバとの接続情報
- 知覚情報
- エラーメッセージ

4.2　RoboCup 再設計（RoboCup 流れ）

クライアントにおける動作の流れを図に示す．

図 4.2 において，まずサーバからのデータを受信し，その受信データをデータの種類（３種類）により異なった動作を実現するために，判別する．この３種のデータについては，後で述べる．

データの種類を判別した後，その情報の加工を行う．これは，各情報は一繋がりのデータで様々な情報を含んでいるため，まずデータを切り離し，それぞれの値に応じた型に格納する作業である．また，ここで，データは後の段階で使用が容易な形に，演算し格納を行う．

4.2　RoboCup 再設計（RoboCup 流れ）

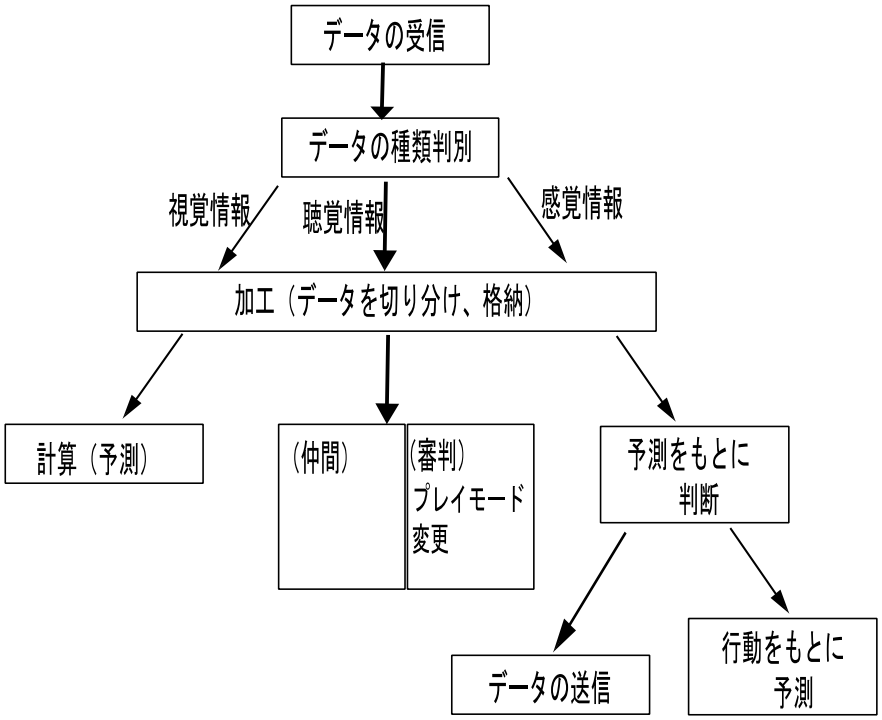

図 4.2　クライアントにおける処理の流れ

この後の段階では，受け取ったデータにより異なった作業を行う．まず，受け取ったデータが視覚情報だった場合，加工したデータをもとに，自プレイヤーの座標位置などを計算により求める．次に，聴覚情報だった場合，聴覚情報には，同チームプレイヤーからのメッセージと，審判からのメッセージの２種類あり，これに応じ，同チームプレイヤーからのメッセージの場合は，そのメッセージに応じた動作を行い，審判からのメッセージは，プレイモードの変更を知らせるメッセージのみのため，それに従いプレイモードの変更を行う．そして，感覚情報だった場合，視覚情報を用いた計算データをもとに，判断を行う．そして，ここでの判断に基づき，プレイヤーは次の行動を行うことになるが，この行動を示すデータをサーバに送信する．また，次なる情報がサーバ側から十分に送信されなかった場合や，次の行動迅速に行える準備とし，ここで行った行動をもとに，行動を行った後のプレイヤーの速度，座標位置，各オブジェクトとの相対位置などを予測，演算する．

このサッカーシミュレーションでは，いかに情報を的確に操作し，ある戦略に基づいた正確な行動を行うかが勝負の決め手となる．また，正確で，かつ的確な行動を行うためには，

「予測」の部分が最も重要な部分となる．

4.3　知覚情報 3 種

この 3 種類のデータは，上記サーバとの通信内容における「知覚情報」とであり,「視覚情報」,「聴覚情報」,「感覚情報」である．これらは情報の種類を示すラベルと共に，それぞれ様々なメッセージフォーマットを持ちサーバからクライアントに送られる．情報の種類を判別するラベルは，視覚情報は see，聴覚情報は hear，感覚情報は sense_body である．視覚情報とは，プレイヤーの視野内において，そのプレイヤーに見えている情報であり，対戦チームのプレイヤーや同チームプレイヤー，ボールやゴール，そして視覚情報を援助するためにコート周辺に配置されている Flag などの「オブジェクト」について情報である。視覚情報のデータフォーマットは，そのオブジェクト名，そのオブジェクトとの相対距離，相対的な方向などの値により構成されている．この視覚情報を用い，プレイヤー自身とオブジェクトに対する相対的な位置関係などを計算することにより，プレイヤーの位置や方向などを求めることができる．

次に，聴覚情報とは，審判，又は，同チームプレイヤーからのメッセージである．審判からのメッセージは決められており，その情報に従いプレイモードなどの変更を行う．また，同チームプレイヤーからのメッセージは，各チームで任意に予め用意することのできるメッセージから選択し送られてくる．そして，感覚情報は，プレイヤーの速度や，スタミナの情報である．

4.4　1 試合

前後半それぞれ 3000 カウントで 1 試合は 6000 カウント（実時間にして約 10 分）の形式で行われるのだが，その中でサーバとの情報の受送信は 100ms の単位で行われる．知覚情報の中で，感覚情報はこの 100ms 毎に必ず送信されるが，視覚情報においては，この感覚情報とは非同期のサイクルである．以下に図を提示する．

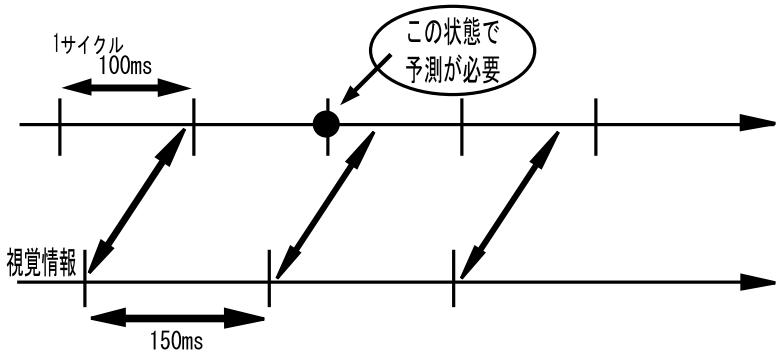

図 4.3　知覚情報との時間関係図

4.5　RoboCup 公理的設計法による設計

ここでは，公理的設計法に基づく RoboCup の設計結果を示す．

4.5.1　FR の定義とマッピングによる DP の設定

CA は,「RoboCup サッカーにおけるクライアントプログラムの作成」である．

次に，FR の定義だが，考えられる最高レベルの機能としては,「サッカーサーバとのデータの送受信」とクライアントプログラムの核となる部分である「考える」機能である．ここで，式（4.1）のように FR を定義すると，

$$\begin{cases} FR1 = \text{データの送受信} \\ FR2 = \text{考える} \end{cases} \tag{4.1}$$

これに対し考えられる DP のセットは，

$$\begin{cases} DP1 = \text{サッカーサーバとの通信データ} \\ DP2 = \text{知覚情報} \end{cases} \tag{4.2}$$

となり，この式（4.1）と式（4.2）を設計行列により確認すると，

$$\begin{Bmatrix} FR1 \\ FR2 \end{Bmatrix} = \begin{bmatrix} X & X \\ X & X \end{bmatrix} \begin{Bmatrix} DP1 \\ DP2 \end{Bmatrix} \tag{4.3}$$

式（4.3）のように，FR1 と FR2 は機能的に相互依存しており，干渉設計となってしまう．ここで，系 1「ある設計において，FR が干渉すなわち相互依存する場合は解の分解を行

え」[*1]に基づき，式（4.4）のように FR を定義する．

$$\begin{cases} FR1 = \text{データの受信} \\ FR2 = \text{考える} \\ FR3 = \text{行動（データ）の送信} \end{cases} \tag{4.4}$$

これに対し考えられる DP のセットは，

$$\begin{cases} DP1 = \text{サッカーサーバからの送信} \\ DP2 = \text{知覚情報} \\ DP3 = \text{決定された行動（サッカーサーバへの送信データ）} \end{cases} \tag{4.5}$$

となり，この式（4.4）と式（4.5）を設計行列により確認すると，

$$\begin{Bmatrix} FR1 \\ FR2 \\ FR3 \end{Bmatrix} = \begin{bmatrix} X & 0 & 0 \\ X & X & 0 \\ 0 & X & X \end{bmatrix} \begin{Bmatrix} DP1 \\ DP2 \\ DP3 \end{Bmatrix} \tag{4.6}$$

式（4.1）の $FR1 =$ データの送受信 の通信部分を式（4.4）のように $FR1 =$ データの受信，$FR3 =$ データの送信 と解の分解を行うことで，準独立設計式（4.6）することができる．ここで，準独立設計は容認できる設計解のため，次のレベルへと分解を進める．ここから先のレベルでは，クライアントプログラムにおいて最も重要な「$FR2 =$ 考える」の設計を示す．

$$\begin{cases} FR21 = \text{加工} \\ FR22 = \text{計算} \\ FR23 = \text{判断} \\ FR24 = \text{予測} \end{cases} \tag{4.7}$$

これに対し考えられる DP のセットは，

$$\begin{cases} DP21 = \text{受信データ} \\ DP22 = \text{加工済データ} \\ DP23 = \text{計算値全て} \\ DP24 = \text{行動決定後の結果データ} \end{cases} \tag{4.8}$$

となり，この式（4.7）と式（4.8）設計行列により確認すると，

$$\begin{Bmatrix} FR21 \\ FR22 \\ FR23 \\ FR24 \end{Bmatrix} = \begin{bmatrix} X & 0 & 0 & 0 \\ X & X & 0 & 0 \\ 0 & X & X & 0 \\ 0 & X & X & X \end{bmatrix} \begin{Bmatrix} DP21 \\ DP22 \\ DP23 \\ DP24 \end{Bmatrix} \tag{4.9}$$

[*1] 付録 A（系 1）参照

準独立設計式（4.9）となるので，更に下位レベルの分解を行う.「$FR21$ = 加工」から「$FR24$ = 予測」までの分解と設計行列による確認を行う.

$$\begin{cases} FR211 = \text{分割} \\ FR212 = \text{格納} \end{cases} \tag{4.10}$$

ここで，$FR211$ = 分割 とは，サーバから受信する知覚情報が，各知覚情報ごとに一繋がりのデータフォーマットであるため,「()」や「スペース」などを目安に分割する機能である. また，$FR212$ = 格納 は，切り分けられたデータを後に使用する変数に格納する機能である．次に式（4.10）に対する DP のセットは,

$$\begin{cases} DP211 = \text{全ての受信データ（知覚情報）} \\ DP212 = \text{分割済のデータ} \end{cases} \tag{4.11}$$

となり，この式（4.10）と式（4.11）を設計行列により確認すると，準独立設計となる．式 (4.12)

$$\begin{Bmatrix} FR211 \\ FR212 \end{Bmatrix} = \begin{bmatrix} X & 0 \\ X & X \end{bmatrix} \begin{Bmatrix} DP212 \\ DP212 \end{Bmatrix} \tag{4.12}$$

「$FR23$ = 判断」の分解

$$\begin{cases} FR231 = \text{検索} \\ FR232 = \text{行動の決定} \end{cases} \tag{4.13}$$

ここで，$FR231$ = 検索 とは，$FR22$ = 計算 のデータから最新データを検索する機能である．また，$FR232$ = 行動の決定 は，行動の決定を行う部分であり，各エージェント（各プレイヤー）に依存し，以下の構造が異なる．次に式（4.13）に対する DP のセットは,

$$\begin{cases} DP221 = \text{過去の計算データ} \\ DP222 = \text{最新のデータ} \end{cases} \tag{4.14}$$

となり，この式（4.13）と式（4.14）を設計行列により確認すると，準独立設計となる．式 (4.15)

$$\begin{Bmatrix} FR231 \\ FR232 \end{Bmatrix} = \begin{bmatrix} X & 0 \\ X & X \end{bmatrix} \begin{Bmatrix} DP231 \\ DP232 \end{Bmatrix} \tag{4.15}$$

「$FR24 =$ 予測」の分解

$$\begin{cases} FR241 = \text{自分の予測} \\ FR242 = \text{環境の予測} \\ FR243 = \text{動くものの予測} \end{cases} \tag{4.16}$$

ここで，$FR241 =$ 自分の予測 とは，2 サイクル前の行動などの情報からからプレイヤーの現在地などを予測する機能である．$FR242 =$ 環境の予測 とは，ゴールやラインなどの動かないものを対象としたプレイヤーとの相対の位置などの計算である．従って，この機能には $FR241$ が前提の機能となる．また，$FR243 =$ 動くものの位置 は，プレイヤー以外の動くもの（ボールや他のプレイヤー）の位置やスピード，プレイヤーの判別などの予測機能である．しかし，ボール位置など重要となる動くものの予測は正確に行うことは，現段階では研究途中である．次に式（4.16）対する DP のセットは，

$$\begin{cases} DP241 = \text{2 サイクル前の行動，最新の視覚情報} \\ DP242 = \text{自分の予測結果} \\ DP243 = \text{2 サイクル前のボール情報（動き），最新の視覚情報} \end{cases} \tag{4.17}$$

となり，この式（efeq:robo-fr24）と式（4.17）を設計行列により確認すると，準独立設計となる．式（4.18）

$$\begin{Bmatrix} FR241 \\ FR242 \\ FR243 \end{Bmatrix} = \begin{bmatrix} X & 0 & 0 \\ X & X & 0 \\ X & 0 & X \end{bmatrix} \begin{Bmatrix} DP241 \\ DP242 \\ DP243 \end{Bmatrix} \tag{4.18}$$

ここまでの分解の階層構造は，図 4.4 のようになる．

4.5.2 Module-Junction 図の作成

前項までの設計行列と階層構造（図 4.4）を元に，Module-Junction 図を作成する．この設計では，独立と干渉の設計はないため，Leaf レベルのモジュールの結合は「c 結合」で示される．Module-Junction 図は図 4.6.2 のような構造となる．

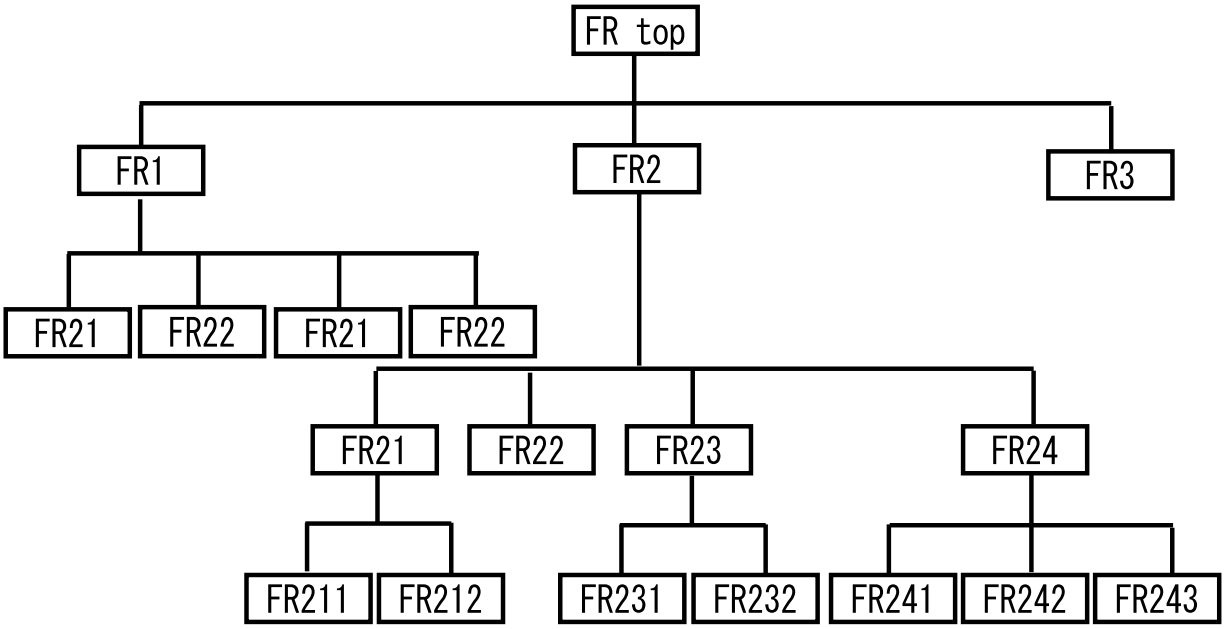

図 4.4　RoboCup,FR 階層構造

4.5.3　フローチャート表現

図 4.6.2 と設計行列，階層構造（図 4.4）を元にフロー図の作成を行う．手順に従い，Leaf レベルの結合から上位のレベルの結合を示し，この作業を反復し，全てのモジュールを結合する（図 4.6.3）．この図により全てのモジュールの関係が確認できる．

4.5.4　Obejct 図の作成と確認

オブジェクト図により，FR と DP の全体の詳細を図により作成する．まず，最上位 FR1 と FR2，FR3 という 2 つに分類する．ここで，最上位 FR は準独立設計のためこの 2 つのモジュールの関係は図 4.7 のような関係になる．

また，2 つに分類した一方の FR1 と FR2 の内部の構造は，以下図 4.8 のようになる．ここでの,「Object」や「d や*」は図 3.3 の対応図に基づく．

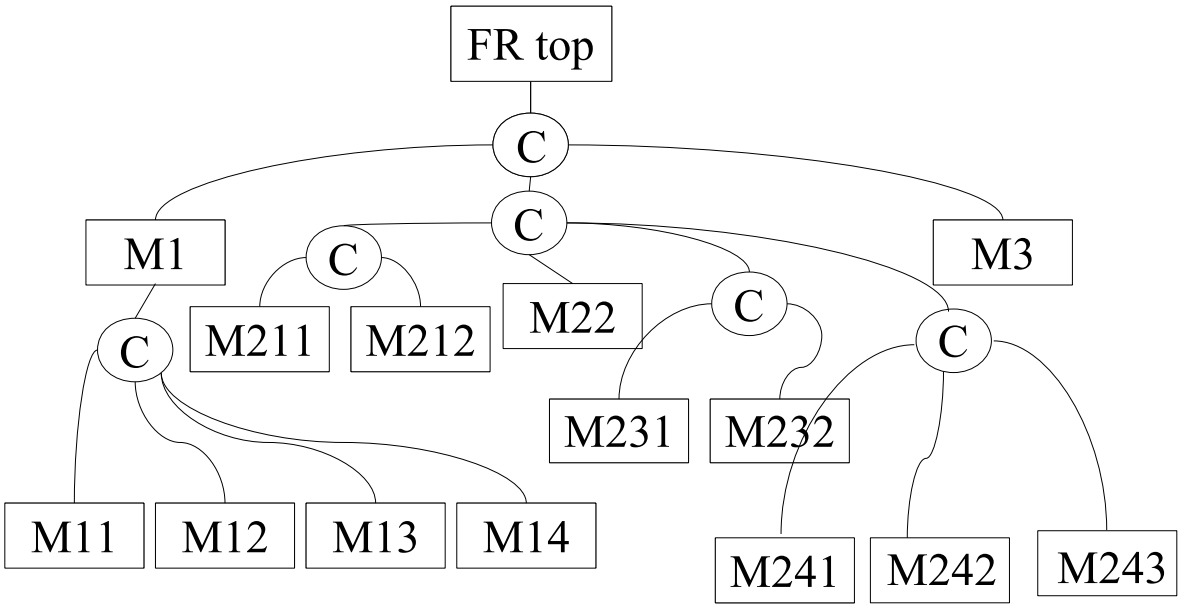

図 4.5　RoboCup，Module-Junction 図

4.5.5　クラス構造の作成

図 4.8 を参照し，クラス図を作成する．

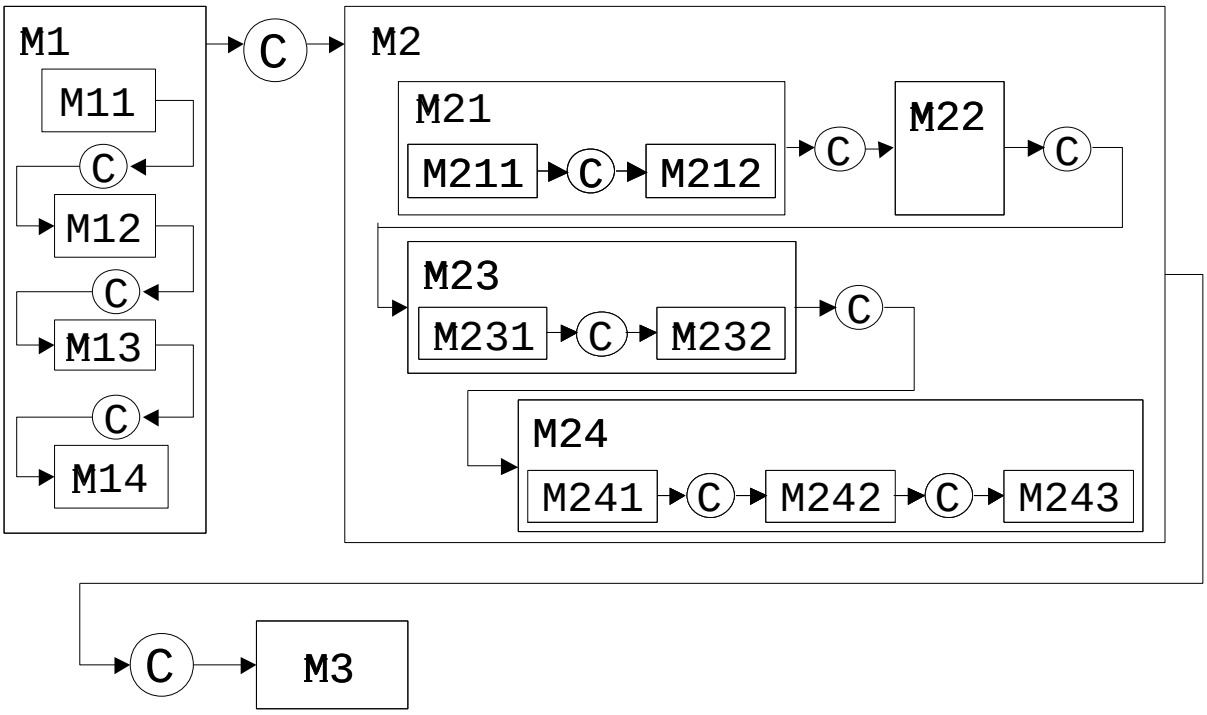

図 4.6　RoboCup, Flowchart 表現

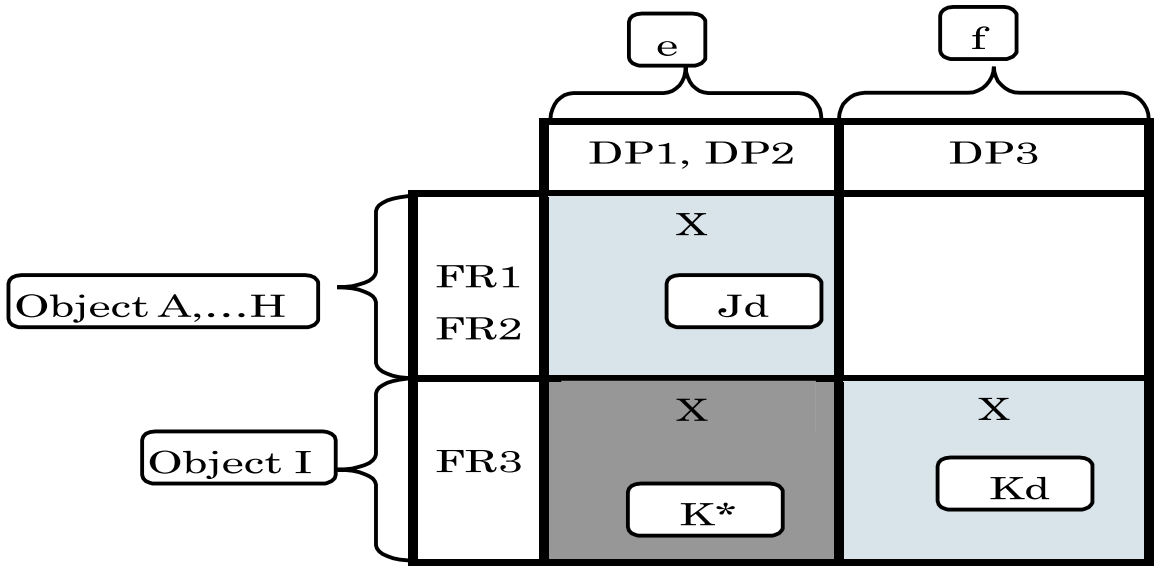

図 4.7　最上位モジュールの関係（オブジェクト図）

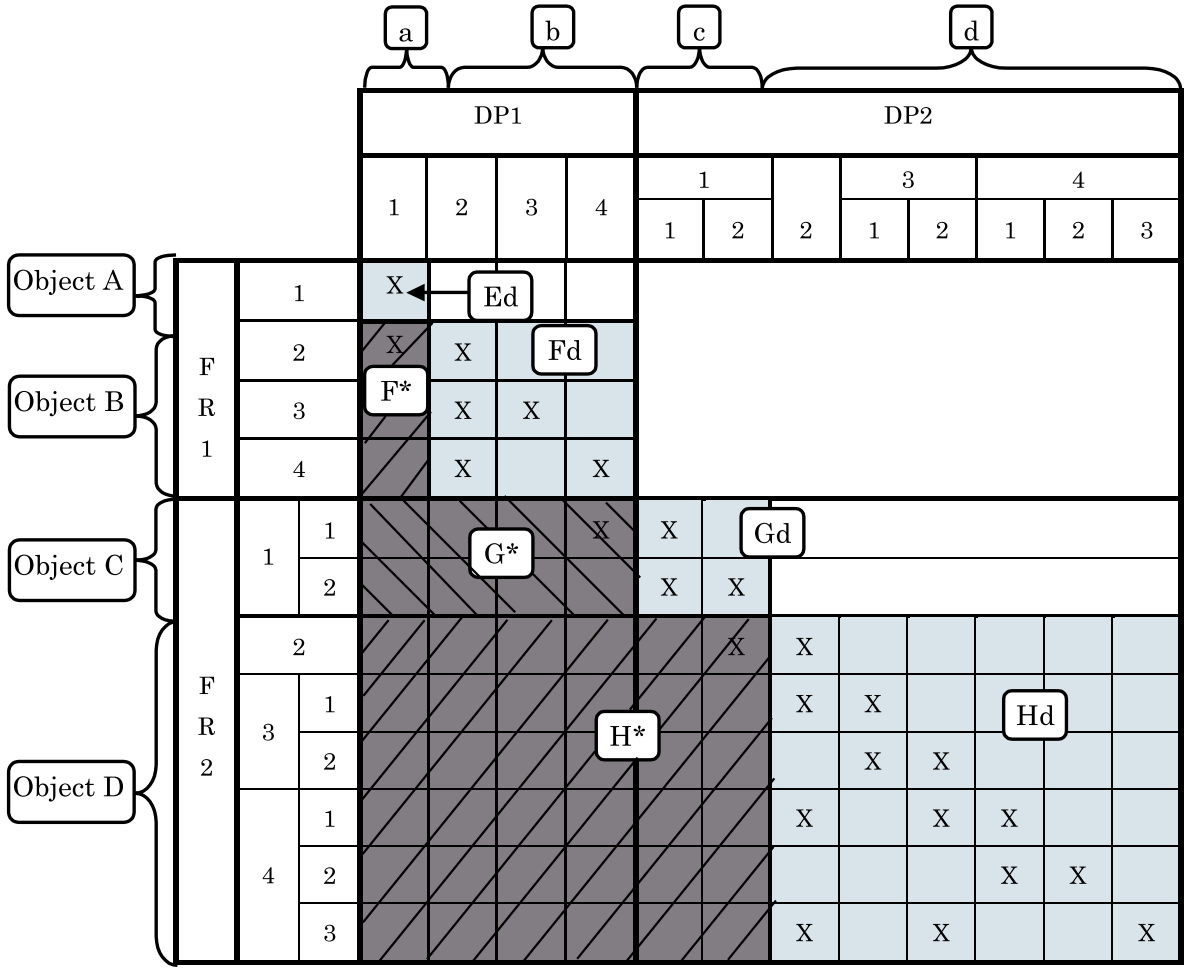

図 4.8　RoboCup，オブジェクト図

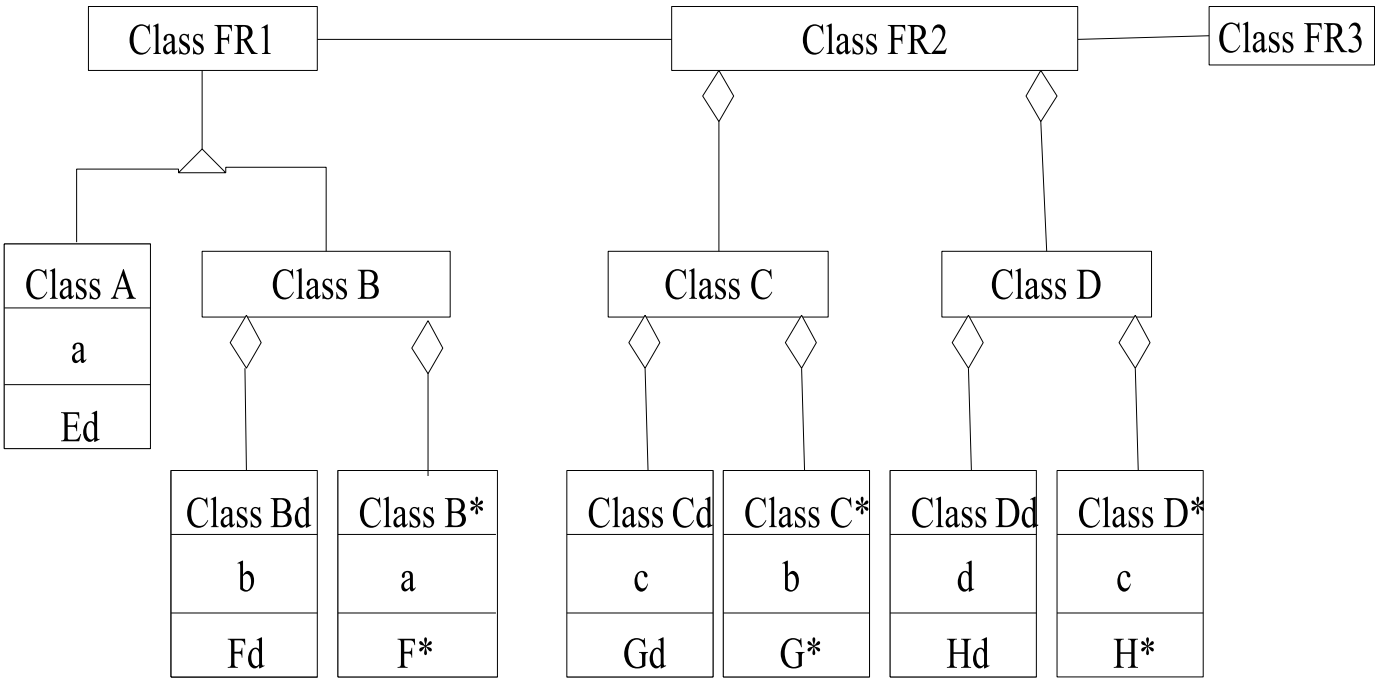

図 4.9　RoboCup, クラス図

第 5 章

結果

この章では，既存プログラムの問題点を挙げ，前章での公理的設計法による設計結果との比較を行う．

5.1　既存プログラムの問題点

既存プログラムの問題点は，

- データのやりとりに無駄がある
- 拡張性がなく，拡張が非常に困難である
- 変更による修正が広範囲に及ぶ

の３点である．

5.1.1　既存プログラムにおける問題点（１）

既存プログラムの問題点として，まず「データのやりとりに無駄がある」という問題点について検証する．下図 4.2 に既存プログラムにおける上位クラス間でどのようにデータが受け渡されているかを示す．ここで,「WorldModel」とは機能は「予測・計算」を行うモジュールである．

図 4.2 において，そのデータの使用を必要とするクラスにデータを受け渡すために，データが介するだけのクラスが存在する.「通信」部で受信されたデータは「加工」によりフォーマットの分割や計算を行う．その後,「WorldModel（予測，計算）」での処理を行うのだが，

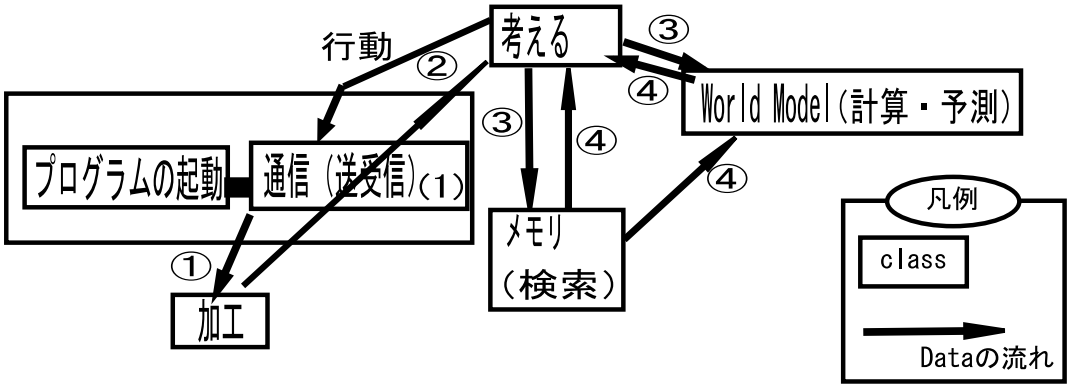

図 5.1　既存プログラムにおけるデータのやりとり

一度「メモリ」を経由しなければ「WorldModel」での処理を行うことができない．そして，その後「考える」部での行動を決定する処理が行われる構造になっている．

この図を元に公理的設計法でのクラス（機能）構造と比較を行う．図 5.2 は，図 4.2 の既存プログラムの構成を公理的設計法で扱われる階層構造により示したものである．また，図 5.3 は公理的設計法の「FR2=考える」の下位層を示す．これら 2 つの図は互いに同一の機能を満たすことを目的としている．既存プログラムの構成名を公理的設計法の設計結果と比較させるため同一の働きをする一部を変更して述べる．ここで変更を行ったのは,「考える，メモリ」部は「判断」,「WorldModel」は「計算・予測」である．

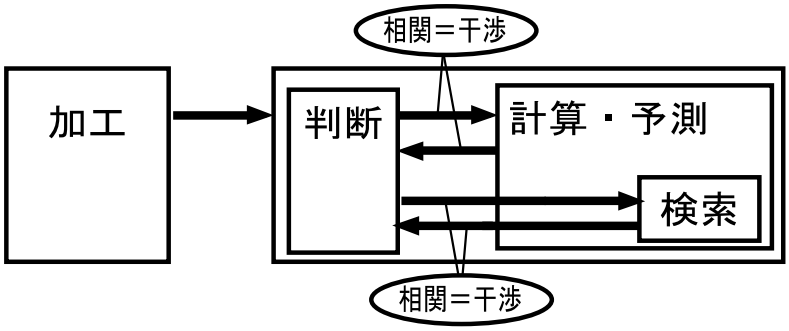

図 5.2　既存プログラムの構成

図 5.2 において,「判断」と「計算・予測」機能間でのデータの相互のやり取りにより，機能が相互依存している．これは公理的設計法に基づく概念では機能干渉が起こっている状態である．これに対し図 5.3 では，前章での設計でも明らかなように機能間は準独立を保って

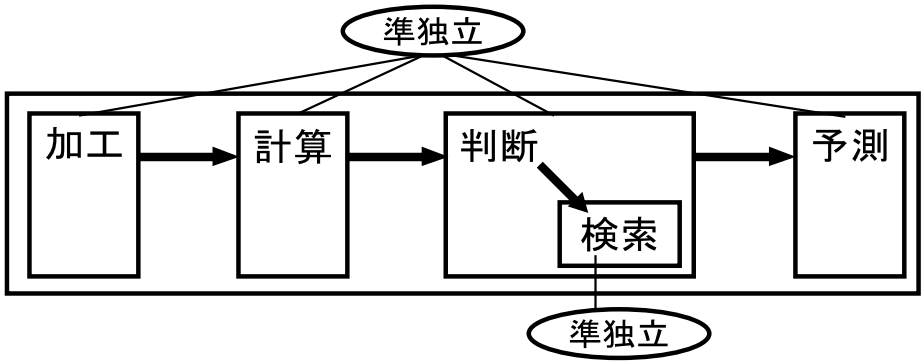

図 5.3　公理的設計法による構成

いる．

ここで，なぜこのようなデータのやり取りの無駄が生じたのかを明らかにする過去のプログラム構成を図（5.4）に示す．

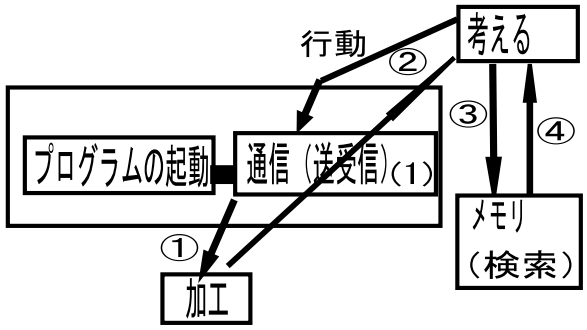

図 5.4　機能追加以前の構造図

これは，「WorldModel（計算・予測）」が追加される以前の状態である．この後「WorldModel」の機能追加を行った．機能追加を行う以前の状態でこの２つの機能にはすでに機能の相互依存が生じ，干渉設計となっていた．その後この干渉設計の機能に同レベルとなる機能の追加を行ったために，更なる複雑な機能依存が生じている．

公理的設計法の観点からすると，このようなデータ遷移の無駄を持つプログラム，つまり干渉が重複している設計は，独立公理に反することはもちろん，情報公理の観点でも容認できない設計解である．

このようなデータ受け渡しの無駄は，既存プログラムにおいて再度に及ぶ無理な機能の

拡張や修正，削除を行ったため生じたと考えられる．機能の拡張や修正の際に，機能実現することを優先とし機能間での独立性維持が考慮されていなかった結果といえる．また，プログラム作成初期段階では，容認されていた構造だった場合でも，その後の拡張や変更の際にデータ依存，つまり機能干渉が起こった可能性がある．

5.1.2　既存プログラムにおける問題点（２）

既存プログラムにおける次の問題点「拡張性がなく，拡張が非常に困難である」について考察する．この拡張性がないという問題は，前項における機能の相互依存による干渉が原因である．様々な機能間で依存関係にあるために新たな機能の追加を行う際に，独立した機能として扱えないためである．また，それまでの設計経緯，設計構造が明白でないために，構造の把握ができないため機能の拡張が困難を引き起こしている．

この問題もまた，公理的設計法による独立性を保ち，設計構造が明らかな設計を行うことで解決できる．例えば，公理的設計法に基づく設計結果に新たな機能追加を行うことを考える．現段階の設計に新たに「記憶（全ての行動の記録）」の機能を追加する．記憶する機能は「行動決定」が前提機能となるため,「考える」が前提機能となる．よって，下図 5.5 のような構造考えられる．

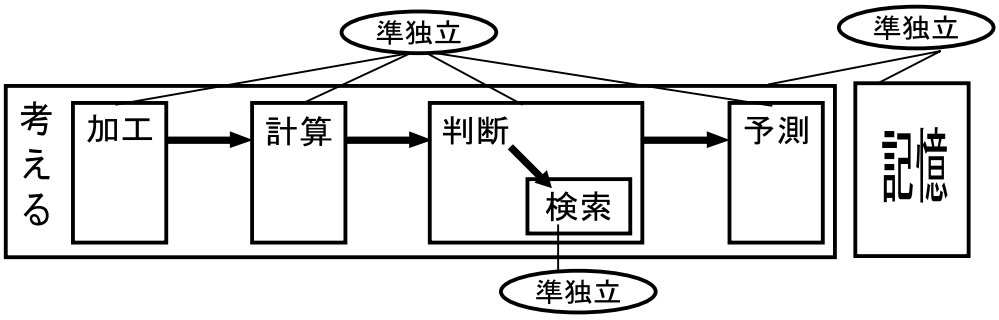

図 5.5　機能の追加後

このレベルに記憶機能を追加しても，設計行列は準独立を保つ．このように，常に独立を保つことで機能の追加も容易に行うことができる．

5.1.3　既存プログラムにおける問題点（３）

既存プログラムにおける３つ目の問題点「変更による修正が広範囲に及ぶ」について考察する．この問題においても既存プログラムが干渉設計であることが原因といえる．公理的設計法による設計では，設計行列やフローチャート表現により，モジュールの関係が明白になっている．しかし，既存プログラムでは，設計実装を行ったものの頭の中での作業となるため，複雑で明白でない．

5.2　まとめ

既存プログラムにおける問題点は，公理的設計法による設計結果と比較することにより，原因追及が行えた．公理的設計法を用いることで，将来の機能拡張や追加なども容易に行うことができる．

しかし，再設計という形ではなく，既存プログラムの修正で干渉の問題を解決することは非常に困難である．公理的設計法では，系１において「ある設計においてFRが干渉すなわち相互依存する場合は，独立化すなわち解の分解を行え」という公式がある．これは，解の分解を行うことで，干渉設計を準独立設計にすることができるというものである．設計初期段階から公理的設計法の概念に基づき設計構造が明らかな場合は，この解の分解も容易に行うことが可能である．しかし，既存プログラムでは，過去の機能追加や変更などの経緯も明白でなく，構造も実装者に依存しているため，公理的設計法の概念に基づく階層構造を作成することは困難である．従って，解の分解を行うことも困難となる．また，プログラムより作られた階層構造に対し，干渉設計を準独立設計にするために解の分解を行うことは，その分解による下位層への影響範囲も明白でない．

公理的設計法の利点は，設計段階での機能干渉を防ぎ，設計の構造が後にまで明白な様々な図として残ることにより，設計後の機能の塚や修正が明白で系統的に行えるという点である．既存RoboCupは，まだまだ研究段階であり，将来様々な機能追加や変更が考えられる．設計ミスによるソフトウェア実行のミス以上に，将来のことを考え，設計解を明白にし

た設計が重要である．このように，公理的設計法は，将来変更や追加が多く考えられるソフトウェアに対し強力な設計ツールといえる．

第 6 章

結論

6.1　公理的設計法による設計の利点

公理的設計法による設計では，設計完了までの経緯，モジュールの相互間の関係などが明白になる．しかし，従来の設計者の考察のみで行われる設計では，将来他の設計者が設計を行う際に，明確な判断材料となる設計の詳細が明白でない．また，この公理的設計法という公式化され系統的である設計法を用いることで，将来受け継ぐ設計者の違いにも柔軟に対応する事が可能である．

公理的設計法による設計の利点は，定式化された設計作業であり，機能の拡張，修正が容易な点である．

6.2　独立化による問題点

本研究では，再設計を行い既存プログラムとの比較を行うことによる検証が目的であったが，その他多くのソフトウェアにおいて，そのソフトウェアにおける干渉の問題を再設計により解決するのは，容易なことではない．既存プログラムを公理的設計法に基づき修正を行うことが最適である．しかし，プログラムの作成経緯など多くの情報が欠落しており，公理的設計法の枠組みに適用することが非常に困難である．また，干渉設計における機能の独立化を行う際，多くの干渉が存在する設計においてどれだけの影響範囲があるのかが明白でない．

公理的設計法におけるソフトウェア設計について影響範囲の不明確な点で同じ事がいえる

6.2 独立化による問題点

ソフトウェアでは，モジュール間で様々なデータのやり取りがあり，多くの機能において機能的に独立ではなく，準独立である場合が多く，機能追加による影響範囲が明白でない．

公理的設計法における今後の課題は，既存ソフトウェアの公理的設計法への適用，及び，追加や機能分解による準独立化での影響範囲について明確に示す定義の提供である．

謝辞

本稿は，著者が 1999 年 7 月より 2001 年 2 月までの高知工科大学工学部情報システム工学科在学中に，同学科ラック研究室において行った研究の成果を記したものである．

本稿における公理的設計法や，本稿作成など種々の書式に至るまでご指導・ご教示を賜った RuckThawonmas 助教授に深く感謝致します．

また，卒業研究活動において，アドバイスを下さった同研究室 RoboCup メンバーの平山純一郎氏，日野慎一氏，三好貴弘氏に深く感謝致します．そして，研究室活動において，他の研究をしているメンバーの方々に種々の面で支えになっていただいたことに感謝致します．

さらに，研究活動中，忙しい中色々な相談にのっていただいきアドバイスをくれた，同学科の戸梶桃さんをはじめとする友人に感謝致します．

最後に，著者が本大学入学時から今までに良い環境を整えて頂いた情報システム工学科の諸先生方に感謝の意を表します．

参考文献

[1] Num Pyo Suh，畑村洋太郎　監訳, “The Principles of Design, 設計の原理–創造的機械設計論”，朝倉書店，1992.

[2] Num Pyo Suh, “Axiomatic Design : Advances and Applications” chapter5:Axiomatic Design of Software, Oxford University Press, 2001 年発売予定.

[3] 井形　弘, “Hiroshi Igata Home Page(Axiomatic Design)”, http://www.sun-inet.or.jp/ igata/.

[4] Jason D.Hintersteiner and Amrinder s. Nain, “Integrating Software Into System:An Axiomatic Design Approach”, Proceedings of the 3rd International Conference on Engineering Design and Automation, Vancouver,B.C. Canada., 1999.

[5] “Soccerserver Manual（日本語版）Ver5.00”, http://www.ita.tutkie,tut.ac.jp/ watta/RoboCup/jmanual.html.

[6] “RoboCup 2000 年度 Autumn キャンプ　配布資料”P1-P10, 2000.

付録 A

公理的設計法における
系・定理の一覧

公理的設計法には，２つの公理の他に以下に示すように７つの系と 16 の定理が与えられている．

A.1　系

系１：干渉設計の独立化

ある設計において，FR（必要機能）が干渉すなわち相互依存する場合は，
独立化すなわち解の分解を行え．

系２：FR の最小化

FR と制約条件の数を最小化せよ．

系３：部品の統合

ある解において FR が独立に満たされるならば，設計の特性を１つの部品に統合せよ．

系４：標準の利用

FR と制約条件を満たすかぎり，標準部品すなわち代替可能な部品を使用せよ．

系５：対称性の利用

FR と制約条件を満たすかぎり，対称な形状-要素を使用せよ．

系６：許容範囲の最大化

FR を設定する場合，許容範囲を可能なかぎり大きくせよ．

系７：情報量の少ない独立設計

一組の FR を満たすために，干渉設計よりも必要情報量の少ない独立設計を探せ．

A.2　定理

定理１：DP 数の不足による干渉

DP（設計変数）の数が FR（必要機能）よりも少ない場合，

それは干渉設計となるか，満たされない FR が生じる．

定理２：干渉設計の独立化

FR が DP よりも多い（つまり $m > n$）のために干渉設計となっている場合，

設計行列の $n * n$ の成分を含む小行列が三角行列ならば，

FR と DP の数が等しくなるように新しい DP を加えることで独立化することができる．

定理３：冗長な設計

DP の数が FR よりも多い場合，それは冗長な設計または干渉設計のいずれかである．

定理４：理想的な設計

理想的な設計では DP と FR の数は等しい

定理５：新たな設計の必要性

ある一組の FR が，新たな FR を加えたり，FR のうち１つを新しいものと取り替えたり，全く異なる組み合わせの FR を選択したりして変更された場合，元の DP による設計解は新しい FR を満たすことはできない．従って新しい設計解を探さなければならない．

定理６：独立設計の経路に対する非依存性

独立設計の情報量は，与えられた FR を満たすように DP を変更する際の順序に依存しない．

定理７：干渉設計・準独立設計の経路に対する依存性

干渉設計・準独立設計の情報量は，DP を変更する順序と，DP 変更の経路に依存する．

定理８：独立性と許容範囲

設計者が規定した許容範囲が次の値よりも大きいならば，それは独立設計といえる．

$$\left(\sum_{j \neq i, j=1}^{n} (\partial FR_i / \partial DP_j) \Delta DP_j) \right) \tag{A.1}$$

の場合，設計行列の非対角成分は，設計検討において無視することができる．

定理９：製作性を考えた設計

製品が製作可能であるためには，製品の設計行列 [A]（製品の FR ベクトルを製品の DP ベクトルに関係付けるもの）と，製作プロセスの設計行列 [B]（DP ベクトルを製作プロセスの PV（プロセス変数）ベクトルに関係付けるもの）の積は対角行列または三角行列にならなければならない．従って，この設計行列のいずれか，つまり [A] または [B] が干渉設計であれば，その製品は製作不可能である．

定理１０：直角率・平行率のモジュール性

設計行列 [DM] を正方小行列に分けることができ，対角にならんだ小行列だけが零行列でないとする．このとき，[DM] の直角率と平行率は，対角線上の各小行列の直角率・平行率の積に等しい．

定理１１：直角率・平行率の不変性

FR ベクトルと DP ベクトルの成分の順序を入れ換えても各対応関係が維持されるならば，設計行列 [DM] の直角率と平行率は不変である．

定理１２：情報量の総和

いくつかの事象についての情報量の総和もまた情報量である．ただし，事象が統計学的に独立でない場合には適切な条件付き確率を用いること．

定理１３：システム全体の情報量

各 FR が他の FR と独立と見なせるならば，システム全体の情報量は，満たすべき FR に関連する各事象の情報量の総和に等しい．

定理１４：干渉設計と独立設計の情報量

機能領域において FR がある状態から別の状態に変化する場合，この変化のために必要な情報量は干渉プロセスの方が独立プロセスよりも多い．

定理１５：設計－生産のインターフェイス

生産プロセスが製品の FR の独立性を脅かす場合，製品の FR の独立性を維持するためには，製品の設計を変更するか，新しい生産プロセスを設計または採用しなければならない．

定理１６：情報量の対等性

設計課題に関連するすべての情報量は物理量の種類によらず等しく重要であり，

重み付けの関数を適用してはならない．