グループウェア 3−4
（1993. 8. 27）

設計プロセスにおける競合解消について

藤田茂　菅原研次

千葉工業大学情報工学科

現在行われている設計作業の多くは、大規模定型設計であるといわれている。設計対象の問題の規模が大きくなるに従い、定型設計であっても設計プロセス間の相互作用の複雑性は飛躍的に増大し、これが設計プロセス全体の効率の低下につながる。この相互作用が発生する大きな原因はプロセス間のさまざまな競合であり、これをさけるためには設計プロセス全体の競合を管理し、それらの競合の効果的解消を機械的に支援する必要がある。本稿では大規模定型設計プロセスのモデルとその相互作用の多くの部分を占める設計プロセス間の競合解消について述べる。

キーワード
競合解消　相互作用プロトコル　大規模定型設計　ソフトウェアプロセス

Conflict Resolution In A Design Process

Shigeru Fujita, Kenji Sugawara

Dept. Computer Science
Chiba Institute of Technology

Most design tasks in the industrial world can be categorized into a class of large-scale routine design tasks. The growth of complexity of interactions among design processes, makes efficiency of the total design task lower. Conflict among design processes is main factor which causes complicated interactions among them. In order to resolve this disadvantage, conflict among design processes should be managed in a total design process, and they should be resolved by a cooperative design support system. In this paper, large-scale routine design processes are formalized into a graph model and a model of conflict resolution is proposed using the process model.

Keywords
conflict resolution, interaction protocol, large-scale routine design, software process

1. はじめに

大規模ソフトウェア開発は多人数の開発チームによる分散開発環境での協調作業により行なわれる。ソフトウェアプロセスについてはこれまで様々なモデルや記述言語が提案されてきた。ソフトウェアプロセスを協調作業ととらえたモデリング[古宮92]や、協調作業をモデル化することによりプロセス再利用を目的とした研究[松浦93]が行なわれている。

ところで現在行われている設計作業の多くは、大規模定型設計作業であるといわれている。すなわち、設計プロセスの多くの部分は既知であり、過去の経験により設計プロセス間の相互作用もほとんどが明示可能であるため、設計プロセス全体をなんらかの記述言語により定式化可能である。しかしながら問題の規模が大きくなるに従い、定型設計であってもプロセス間の相互作用の複雑性は飛躍的に増大し、これが設計プロセス全体の効率の低下につながることが指摘されている。この効率低下をさけるためにはプロセス間相互作用を分析、定式化し、このモデルに基づいて、設計プロセス全体の相互作用を効果的に管理し、機械的に支援しなければならない[Klein 92]。

本稿は大規模定型設計プロセスのモデルとその相互作用の多くの部分を占める設計プロセス間の競合解消のアクティビティを定式化することを目的としている。

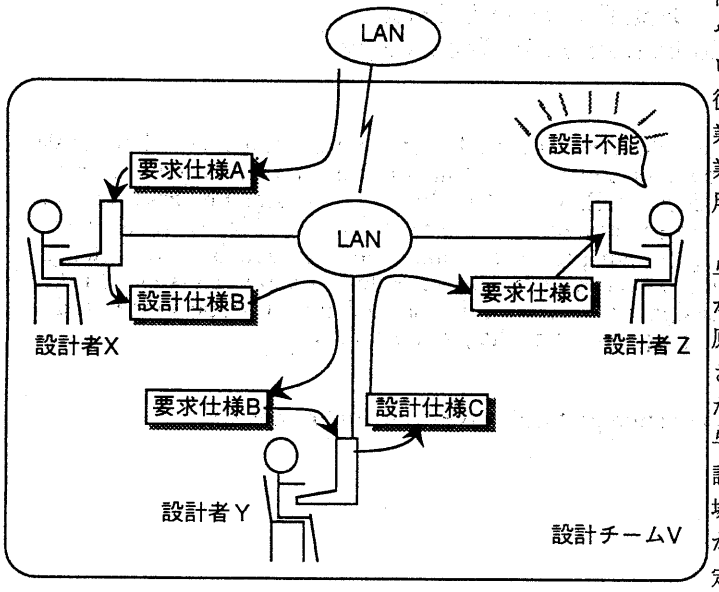


図1　分散開発環境における競合の発見

2節ではソフトウェアプロセスにおける競合の直感的解釈について述べるその競合を解消する手順を定式化する準備として、3節でプロセスモデルの定式化をおこない、4節ではプロセス間相互作用を定式化し、5節でそれらの相互作用を利用して競合解消を行う手順を示す。

2. ソフトウェアプロセスと競合

大規模プロセスであるソフトウェア開発を支援するためには、開発チームの協調作業を効率良く支援することが重要であるが、協調作業を理解するうえで、競合(Conflict)の理解の重要性が指摘されている[Thom93]。競合とは設計作業者間、あるいはより抽象的に述べると設計プロセス間に発生する設計結果の矛盾、概念解釈の相違、意図や目標の衝突、暗黙の、あるいは明示的合意、協約の違反などの相互作用、相互依存、共有目標に関する矛盾、失敗などを意味する。協調作業環境とは、独立した設計者の間に必然的に内包される競合を効率良く解消することを協約し、その意図の基で行動することを前提とした作業環境である。

ところで現実に行なわれている設計作業の多くは定型作業であると言われている[Brown 89]。定型作業とは問題の分割が既知であり、分割された問題の解法が既知と見なされる作業である。定型設計作業では、処理のプロセスが既知であるので、過去の作業経験に基づいてプロセスの作業分担や、作業結果（設計仕様書）の受け渡しや問い合わせの内容のプロセス間プロトコルも基本的に定まっている。従ってプロセスに割り当てられた作業を実行する設計者は定められた作業手順に従い、通常の設計知識を利用しながら設計作業を進めていく。

ところが定型設計作業においても、与えられた設計問題によって、競合が発生することが考えられる。この原因として、その設計チームに想定された問題の範囲に対して与えられた問題の一部がはずれている場合と、与えられた問題の範囲内であっても、設計者の知識不足、誤りなどがある場合が考えられる。このような競合が発生した場合には、その原因を同定し、競合を解消しなければならない。定型作業において、問題のサイ

ズが小さい場合は競合解消は比較的簡単に行なわれるであろう。ところが問題のサイズが大きくなると、プロセス間の相互作用、相互依存は複雑になり、定型設計作業であっても競合解消は困難な問題になる。

例えば図1に示すソフトウェアの分散開発環境の単純な例について考える。設計者X, Y, Zから構成される設計チームVの定型的協調作業の一部を考える。他の設計チームから与えられた要求仕様書は、通常Xにより設計仕様に変換され、Yに対して要求仕様書として与えられ、Yの設計結果はZに対して要求仕様として与えられるものとする。

設計者Xはチームに与えられた設計問題Aを、このチームの定型処理に属する問題であると判断し、定型の作業手順に基づいて処理を行い、設計仕様書Bを作成し、Yも要求仕様書Bを定型問題と判断し、定型の作業手順に基づいて設計仕様書C-を生成する。ところが、その要求仕様書Cを検査した設計者ZはこれはZが処理可能な定型作業の範囲を逸脱しており、かつZの設計知識ではZが満足する解を導き出せないと判断したことにしよう。

設計Zは要求仕様書Cに関する競合が発生したと解釈し、この競合の発生原因を特定これを解消しようとする。この戦略にはいくつか考えられるが、ここでは簡単のため、直前のプロセスを担当している作業者Yとの間の競合と理解するものとする。この場合図2に示すようにZはYに対して再設計要求を出し、Zが作業可能な内容に設計仕様Cを変更することを要求する。Yはその要求を満足するように努力するが、Yへの要求仕様がBのままではZ-の要求に応えられないと判断する。これは競合の伝幡を意味する。その結果YはXに対して再設計要求を出し、もしXが新しい設計要求仕様書B'を生成可能であれば、Zが提起した再設計要求をX, Y, Zの間でコミットし、その結果Zは設計可能な要求仕様C'を得ることができ、Zで発生した競合は解消される。

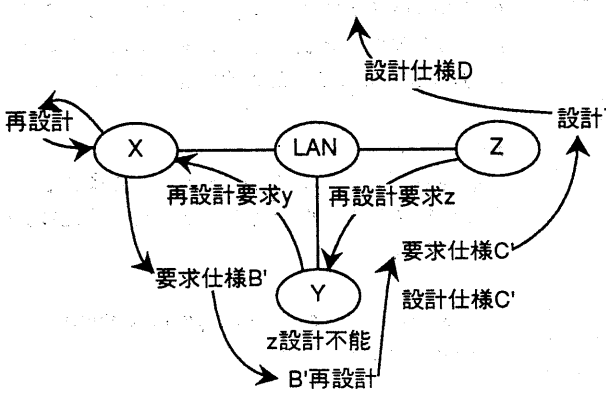


図2　競合解消の例

以上の例は極めて単純化されてた競合解消の例であり、実際はもっと複雑な相互作用をX, Y, Zは持つであろう。或いは他のチームとの競合解消が必要になるかもしれない。いずれにしても大規模プロセスにおいて競合解消をアドホックに行うことは全体の設計効率の低下につながり、従ってこの競合解消の効率的モデル化とそれに基づく効果的支援機構を実現することは重要な問題である。

3.　ソフトウェアプロセスモデル

3.1.　プロセスの基本モデル

本節では競合とその解消を定式化するために必要な基本モデルとして定型ソフトウエアプロセスのモデルを定義する。

ソフトウェアプロセス（以下単にプロセスと呼ぶことにする）の基本タスクとは図3に示すように、与えられる入力プロダクトをある決められた手続きに基づいて出力プロダクトに変換する作業である。

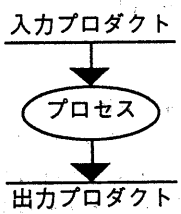


図3　基本アクティビティ

プロダクトとは仕様書、マニュアル等の文書、テストの結果、実行プログラムなどである。設計プロセスにおいては、入力プロダクトを要求仕様、出力プロダクトを設計仕様と呼ぶことにする。

プロセスは以下の5項組で表される。

$$p = \langle ip, op, pre, post, pg \rangle$$

但し、ipは入力プロダクト、opは出力プロダクト、preは入力制約（preconstraint-）と呼ばれipに関する制約条件の記述であり、postは出力制約（postconstraint）と呼ばれopに対する制約条件の記述である。

pgはプロセスグラフと呼ばれプロセスp-の処理過程をさらに詳細に記述した部分プロセスから構成されるグラフである。すなわち図4に示すように、プロセスは階層的に定義され、プロセスグラフpgにより詳細化が行われる。入力制約はプロセスが解き得る問題を限定するためのもので、出力制約はそのプロセスが行う問題解決の品質を明示している。

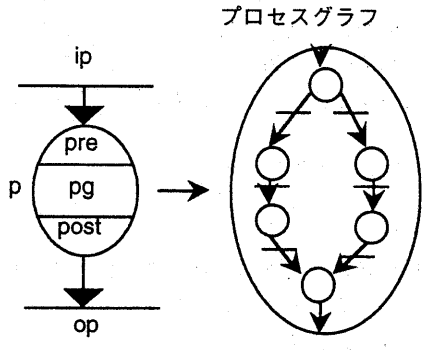


図4　プロセスの階層的定義

プロセスグラフpgは方向付きグラフであり、次のノード集合pgnとアーク集合pgaで記述される。

$$pg = <pgn, pga>$$

但し

$$pgn = pcn \cup pdn$$

であり、pcnはプロセスノード集合と呼ばれ、pdn-はプロダクトノード集合と呼ばれる。pcnは図3.1-にしめされるプロダクトの変換を行うプロセスの集合であり、pdnは図3に示される処理対象のプロダクトの集合である。

アーク集合pgaは2つの部分集合in-arcとout-arcの和集合で表される。

$$pga = \text{in-arc} \cup \text{out-arc}$$

但し

$$\text{in-arc} = \{(n_1, n_2) \mid n_1 \in pdn, n_2 \in pcn\}$$
$$\text{out-arc} = \{(n_1, n_2) \mid n_1 \in pcn, n_2 \in pdn\}$$

dc-arcの要素はプロセスに対する入力アークと呼ばれ、out-arcの要素はプロセスからの出力アークと呼ばれる。プロセスpが内部にプロセスグラフを持たないとき（pg=∅と書く）、そのプロセスは素プロセスと呼ぶ。

3.2　プロダクト

プロセスの処理対象と処理過程であるプロダクトは以下に示すように多数の属性から構成され、属性間には様々な関係あるいは制約が存在する。あるプロダクトの属性およびそれらの関係制約を規定するモデルをドメインモデルという[渡辺93]。ドメインモデルはそれらに属する事例の抽象化されたクラスの集合であり、逆にプロダクトはドメインモデルに定義されたクラスの具象化された事例である。クラスで定義された属性の値を与えることをインスタンシェーションという。その結果生成されたプロダクトを事例（インスタンス）と呼ぶことにする。
以下プロダクトの定義を行う。

prod = <name, domain, generated_from, provided_for, {description}>

但し、nameはプロダクトの識別子、 domainはこのプロダクトが属するドメインの名前である。generated_fromはこのプロダクトが出力されるプロセス名であり、provided_forはこのプロダクトが入力されるプロセス名である。descriptionは仕様記述などのプロダクトの内容であり以下のように定義される。

description = <product_name, description_class, interface, {attribute}>

descriptionはドメインモデルに含まれる、記述の項目を表わすクラスdescription_classの事例であり、項目の値が具体的に書き加えられたものである。attributeは属性名と属性値の2項組である。interface-は他のdescriptionとの関係の記述である。

3.3　制約記述

制約はプロダクトのdescriptionの各属性値に対して付加された条件記述である。条件記述は次のように定義される。

constraint = <const_id, description, attributes, operator, range>

descriptionの属性名attributesに含まれる属性名の

属性値v_1, ..., v_nに対してoperatorを作用させた値operator(v_1, ..., v_n)がrangeに対して

operator(v_1, ..., v_n)⊂ range

を満足するときconstraintは充足するという。range-の表現として不等式あるいは集合などが与えられる。

入力制約pre-constraintは条件記述の集合である。

pre-constraint = <name, product_name, {constraint}>

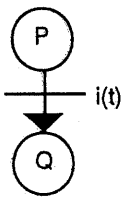
出力記述post-constraintも同様に定義できる。

3.4 設計ポリシー

設計タスクは入力プロダクトドメインに属する事例を出力プロダクトドメインに属する事例に変更する作業である。

あるプロセスが処理可能な入力プロダクトのrangeを規定する条件を入力制約と呼ぶ。このとき処理するプロセスにとって制約条件を厳しくとれば、設計タスクが成功する確率が高くなる。しかし、プロセスが処理可能なプロダクトは減少する。一方、制約をゆるくとれば、設計タスクが失敗する危険率は高くなるが、処理可能なプロダクトの範囲は増大する。

すなわち入力制約の与え方により設計プロセスの性質、能力が異なる。このように設計プロセスの性質、能力に影響を与える要素を設計プロセスの設計ポリシーと呼ぶことにする。すなわち設計プロセスはある設計ポリシーのもとで、入力プロダクトを出力プロダクトに変換する。同じ入力プロダクトにたいしても、設計ポリシーが異なれば、その設計結果である出力プロダクトは一般に異なる。



i(t): 時刻（タイムスタンプ）tに
プロセスPとプロセスQの
間に発生したインタラクション

図5 プロセス間インタラクション

設計ポリシーは一般に以下の項目からなる。

（1）入力制約ポリシー
（2）出力制約ポリシー
（3）設計精度ポリシー
（4）対人間相互作用ポリシー
（5）対プロセス相互作用ポリシー

各ポリシーの値はvery_cheap, cheap, average, expensive, very expensiveなどの段階をとる。

operatorを属性値の直積からrangeへの写像とすると、オペレータ写像OpeMapはポリシーvからoperatorへの写像となる。すなわちOpeMap(v)はあるoperatorである。

同様にポリシーからrangeへの写像RangeMap(v)が定義される。入力制約と出力制約のポリシーが変更されると、それらの条件記述は計算されなおされる。

設計精度ポリシーは、入力プロダクトを出力プロダクトに変更する設計タスクの精度に影響を与える。設計知識はこのポリシーの値に応じて知識の適応のしかたを制御する。対人間ポリシーとは、設計タスクや競合解消などにどれだけ人間の能力に依存するかということであり、値としてquite-dependent, dependent, average, independent, quite-independentの値を持つ。対プロセス相互ポリシーも同様である。

4. プロセス間相互作用

協調作業や競合解消を行うためにプロセスは様々な相互作業をおこなう。相互作用はプロセス間での仕様の受け渡しの関係や依存関係によって発生する。図5にプロセス間相互作用の概念をグラフで表わす。

時刻tでプロセスPとプロセスQの間に発生した相互作用をint(t)と書くことにする。時刻tはタイムスタンプと呼ばれ、相互作用が発生した順に自然数が割り当てられる。

相互作用は大きく2種類に分類される。

(1)プロダクト
　　仕様書、検査結果、プログラムなど
(2)メッセージ
　　要求、通知、依頼など

相互作用として、プロダクトの受け渡しに限定したグラフを基本相互グラフと呼ぶ。図6は基本相

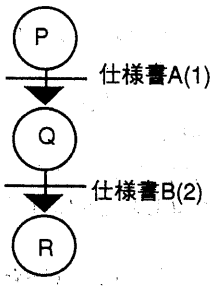


図6　基本インタラクショングラフ

互作用グラフの一部の例である。

基本相互作用グラフから構成されるプロセスグラフを基本プロセスグラフと呼ぶ。基本プロセスグラフは定型プロセスの骨格を決めるグラフであり、定型プロセスにおける競合解消は基本プロセスグラフに付随する拡張グラフとして定義される。

プロセス間相互作用は以下の7種類に分類される。

(1)プロダクトの受け渡し　　Send

(Send process product augment t)

仕様書などのプロダクトの発信を行う。

processは相互作用を行う相手プロセス名、productは受け渡しされるプロダクト、auxiliaryは付加情報、tはタイムスタンプである。

(2)アクション要求　　Request

(Request process action augment t)

相手プロセスに対して様々な要求を行う。

actionは再設計要求、仕様変更要求、情報請求、知識請求、作業委託、設計ポリシ変更要求など、相手プロセスprocessに対する動作要求であり、auxiliaryはその内容を表わす。

(3)応答　　Notify

(Notify process information t)

Requestされた情報、状況などの通知を行う。

informationはRequestにより要求された情報/知識である。processはRequestを出したプロセス名である。

(4)合意　　Commit

(Commit process request t)

Requestされた要求を受諾した旨の通知であり、プロセス間の合意が成立する。

processはRequestを出したプロセスで、requestはこのコミットの対象となったRequestメッサージそのものを返信する。

(5)拒否　　Refuse

(Refuse process request reason t)

Requestを拒否する。

processはRequestを出したプロセスで、requestは拒否の対象となったRequestメッセージそのもの、reasonは拒否理由である。

(6)逆提案　　Propose

(Propose process request propoal t)

Requestを受諾はできないが、受諾可能なRequestを相手に逆提案する。あるいは相手のProposeに対して、受諾可能なProposeを相手に逆提案する。processは相手プロセスであり、requestはproposeの対象となるRequestメッセージあるいはProposeメッセージである。proposeは逆提案の内容である。

(7)取消し　　Cancel

(Cancel process calcel t)

Requestを取り消す。

Commitが成立していないRequestを取り消す。processはCancelを出したプロセス、calcelはRequestした内容そのものである。

上記7種類の相互作用は、その作用の順序がプロセス間プロトコルとして規定されている。その典型的パターンを図7に示す。

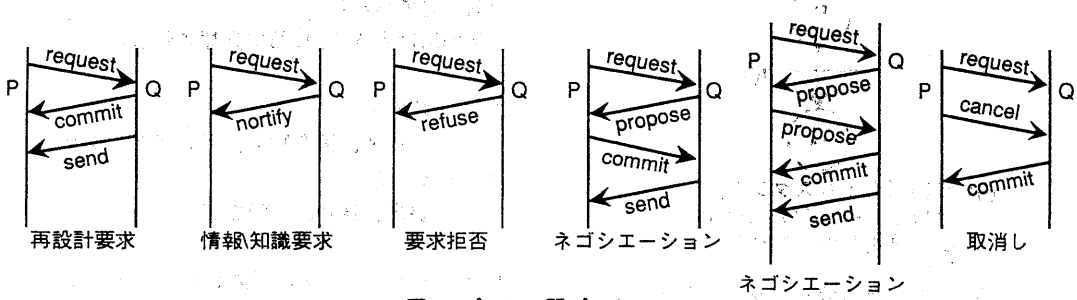


図7　プロセス間プロトコル

5. 相互作用による競合解消モデル

5.1 競合解消手順

分散開発環境における設計プロセスの設計失敗あるいは設計不能の事象が発生する原因には以下のことが考えられる。

(1)設計結果に関するプロセス間競合
(2)設計ポリシーに関するプロセス間競合
(3)プロセス間資源競合
(4)制約記述の誤り
(5)設計知識の誤り、欠如
(6)競合解消戦略のプロセス間競合

大規模設計プロセスにおいては、特に(1) (2) (3)の問題の解決が重要である。図8に各設計プロセスが行う競合解消の手順の概要を示す。

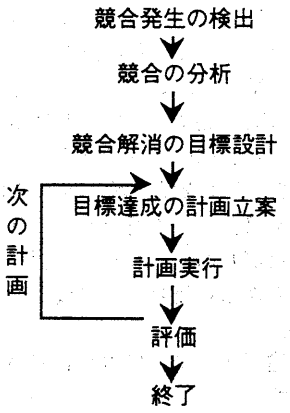


図8 競合解消の手順

5.2 競合の検出と分析

各プロセスは入力制約と出力制約を持ち、いずれかの制約が違反した時、そのプロセスが関係する競合が検出される。各プロセスは入力制約違反テーブルと出力違反テーブルを持つ。違反テーブルとは、違反の原因とそれを解決するための目標を記述した表である。制約違反が発生した時、プロダクト、違反項目、設計活動履歴、設計ポリシーなどから、原因に関する可能性の優先順位を決定し、それから競合解消を実行する目標の優先順位を決定する。

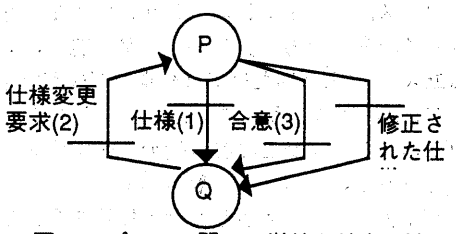


図9 2プロセス間での単純な競合解消

入力制約違反テーブ

原因	目標	計画
(ic1)前プロセスの設計結果と対象プロセスの入力制約との競合	(ig1)前プロセスの仕様変更。ポリシー不変	(ip1)・前プロセスに再設計要求 ・合意待ち
(ic2)対象プロセスの入力制約記述	(ig2)入力制約記述修正。(ポリシー不変)	(ip2)・設計を強行して出力制約を充足することの確認。 ・要求仕様を充足可能とするよう入力制約を修正
(ic3)設計ポリシーの競合	(ig3)入力プロダクトを受理可能とするようポリシーの変更を行う。	(ip3)・入力プロダクトを受理可能とする設計ポリシーを探す ・その設計ポリシーで矛盾が生じないかの確認

図10 違反テーブル

5.3 競合解消の相互作用系列の計画と実行

制約違反に対して、分析の結果定められた目標の優先順位に従って、目標達成のための相互作用の系列を立案する。この系列は目標達成のための暫定的な行動予測であり、相互作用の過程で修正を行いながら目標の達成を試みる。各プロセスは例えば図9のような競合解消のための典型的相互作用系列を部品として所有し、これを状況に応じて修正しながら組み合わせて再利用してゆく。目標が達成できない場合は次の優先順位の目標に切り替える。

図11に違反テーブルに書かれている目標と計画の例を示す。

5.4 競合の伝幡と競合解消活動の競合

競合解消の基本動作は図9に示すような、2つのプロセス間でのネゴシエーションによる合意の形成とその実行である。しかしながら競合によっては2つのプロセス間だけでは合意の形成ができず、図2に示した例のように、他のプロセスの間にネゴシエーションなどの競合活動が伝幡することが考えられる。このとき、競合解消のための相互作用が輻輳したり、矛盾した相互作用を立案して、競合解消活動自体の競合が発生する可能性がある。これを分散的に管理し、処理することはかなり困難な作業であるため、各プロセスグループに管理プロセスをおき、プロセス間相互作用を集中的に管理する方式が考えられる。

6. おわりに

分散開発環境で大規模定型ソフトウェア開発を支援するうえで、設計プロセス間の協調及び競合の解消を支援することはプロセス全体の効率を向上させるうえで重要である。本稿では競合解消を支援するためのメカニズムを分散開発環境に構築するための第一ステップとして大規模定型プロセスの定式化を行い、このモデルに基づいて競合の発生とプロセス間の相互作用を定義し、これを用いて競合解消の手順を記述する方式を示した。このプロセスモデルの実行モデルとしてマルチエージェントモデルに基づく競合解消の実行モデルが提案されている[藤田93]。また設計ポリシーという概念を提案し、これにより競合が発生し、ポリシーを協調的に変更することにより競合が解消できる場合もあることを指摘した。

謝辞

本研究の一部は平成5年度千葉工業大学研究所受託研究（沖電気工業（株））の補助をうけている。日頃熱心な討論と有益な助言を頂く、沖電気工業（株）総合システム研究所の木下哲男博士に深謝する。

[参考文献]

[Klein 92] M. Klein, "Conflict Management in Cooperative Design Teams", Working paper of The 11th Inter. Work. on D.A.I, 1992

[Tom 93] P. Thomas and J. Riddick, "Organizational Structures, Computer Supported CooperationWork and Conflict" in CSCW: Cooperation or Conflict, S. Easterbook (Ed.), Spring-Verlarg, 1993

[松浦93] 松浦、本位田 "ソフトウェアプロセスにおける協調とその抽象化について"、コンピュータソフトウェア、Vol. 10, No.2, pp.48-64, 1993, pp.48-64

[古宮92] 古宮、"ソフトウェア協調型設計過程のモデル化と知的支援方式について"、情報処理学会グループウェア研究会資料、1992年9月

[渡辺93] 渡辺、菅沼、菅原、木下、白鳥、"応用層プロトコル設計のためのドメイン分析とその知識表現"、電子情報通信学会人工知能と知識処理研究会資料、1993年7月

[藤田93] 藤田、菅原、"並行設計プロセスのためのマルチエージェント指向モデル"、電子情報通信学会人工知能と知識処理研究会資料、1993年7月