グループウェアと
ネットワークサービス　43−10
（２００２．３．２３）

サービスルールの追加による異常状態とデッドロックの動的回避法と評価

笠間 嘉宏，井上 伸二，角田 良明
広島市立大学 情報科学部 情報工学科

近年、電話網に代表されるネットワークにおいて新しいサービスの追加に伴いサービス同士の機能の競合が起こっている．この競合をフィーチャインタラクションと呼ぶ．現在，フィーチャインタラクションは競合を起こすサービスに同時に加入させないことで，静的に解消している．しかし，この静的解消法ではユーザに提供できるサービスに制限がかかってしまうため，サービスの動作中にフィーチャインタラクションを検出，解消する動的回避法が提案されている．しかし，動的回避法もユーザに対するイベントの制限やデッドロックといった問題を抱えている．本稿では，動的回避法の改善法を提案し，実験によって有効性を確認した結果を示す．

A Dynamic Avoidance Method for Abnormal States and Deadlocks and Its Evaluation

Yoshihiro KASAMA, Shinji INOUE, Yoshiaki KAKUDA
Department of Computer Engineering, Faculty of Information Sciences, Hiroshima City University

In recent years, new service is developed and added in the communication network. As a result, the phenomenon in which the functions of new services and old services cause competition has occurred. This competition is called feature interaction. Currentry, feature interactions are resolved by the method not simultaneously providing the services which interact with each other. In this method there is dificiency that the provided services are restricted. In order to cope with this dificiency, a dynamic method for avoiding abnormal states, a typical feature interaction, during operation of services is proposed. However, in practice, restriction of the event execution is difficult, and in some case system states fall into deadlocks. Therefore, this paper proposes an improved method which avoids abnormal states and deadlocks. In order to evaluate the proposed method, this paper also shows experimental results using a simulator.

1　はじめに

近年, 電話網に代表されるネットワークにおいてユーザの要求の多様化にともないサービスも多様化している．これらの複数のサービスに加入した際に各サービスの機能が競合して不具合が起こっている．この不具合のことをフィーチャインタラクションと呼ぶ [1]．現在フィーチャインタラクションは競合を起こすサービス同士を同時に使用させないことで静的に解消している．静的解消法は，フィーチャインタラクションを引き起こすサービスの両立を排除することでその危険性を除外するものである．しかし, この静的解消法ではユーザが希望するサービスであっても, 既存のサービスとフィーチャインタラクションを起こす可能性があればどちらかのサービスしかユーザに提供できないという欠点がある．またサービスの数が増えるとフィーチャインタラクションの有無を調べる事が困難になってしまう．

この欠点を補うためにサービスの動作中に競合を回避しようとする動的回避法が提案されている [2]．しかし現段階では動的回避法はユーザに対する実行イベントの制限やデッドロックという問題を抱えている．本稿では, これらの問題に対する解決法を提案する．

2　サービスの記述

2.1　サービス仕様

サービスの機能を形式的に表したものをサービス仕様と呼ぶ. 本稿では, サービス仕様としてルールベールサービス仕様を採用する. このルールベースサービス仕様では 1 つのサービスをサービスルールの集合で表す. サービスルールとはそのサービスにおける 1 つの機能を表すものであり以下の型式で記述される.

前条件 [イベント] 後条件

ここで前条件とはどういう時にイベントを実行できるかを表すもので, 後条件とはそのイベントにより前条件がどのように変化するのかを表している. また前条件, イベント, 後条件は電話網と各ユーザがどうなっているのかを示す述語で表される. 述語は複数のパラメータを持ち第 1 パラメータで表されるユーザの状態を示すものである. 例えば $\mathrm{path}(a,b)$ はユーザ a はユーザ b と通話中であることを示している.

例として，代表的な電話サービスである三者通話 (以降，TWC と略す) と割り込み電話 (以降，CW と略す) のサービスルールの一例を，それぞれ例 1 及び例 2 に示す.

例 1 :$\mathrm{twc1}(a,b)$,$\mathrm{twc2}(a,c)$,$\mathrm{path}(a,b)$,$\mathrm{path}(a,c)$
$[\mathrm{onhook}(a)]$ $\mathrm{idle}(a)$

例 2 :$\mathrm{cw}(a)$,$\mathrm{hold}(a,b)$,$\mathrm{path}(a,c)$ $[\mathrm{onhook}(a)]$
$\mathrm{cw}(a)$,$\mathrm{hold}(a,b)$,$\mathrm{ringing}(a,b)$

TWC は，加入者が通話中に第三者に呼を要求し，三者同時に通話ができるサービスである．例 1 の前条件は，$\mathrm{path}(a,b)$ と $\mathrm{path}(a,c)$ によってユーザ変数 a と b およびユーザ変数 a と c が通話中であることを示している．そして $\mathrm{twc1}(a,b)$ と $\mathrm{twc2}(a,c)$ によってそれらが三者同時の通話であることも示している．例 1 のサービスルールは，上述の前条件を表している述語がすべて真であるとき，ユーザ変数 a が受話器を置くことによって発生する $\mathrm{onhook}(a)$ が実行可能であることを示している．そして上述のイベントが実行されたとき，イベント実行後のユーザ変数 a の状態が $\mathrm{idle}(a)$ というアイドル状態に変化することを示している．要するにイベント実行は，前条件で表されている述語を，後条件で表されている述語に置き換えることに相当する.

CW は，加入者が通話中に第三者からの呼を受理し，二者を切り替えて通話できるサービスである．例 2 の前条件は，$\mathrm{hold}(a,b)$ と $\mathrm{path}(a,c)$ によってユーザ変数 a と c が通話中であるとともに，ユーザ変数 a が b との通話を一時的に保留にしていることを示している．そして $\mathrm{cw}(a)$ によってユーザ変数 a が b および c との通話を切り替えながら通話することができることを示している．例 2 のサービスルールはイベントが実行されたとき，イベント実行後のユーザ変数 a の状態は，$\mathrm{path}(a,c)$ がなくなることによりユーザ変数 c との通話が切れ，$\mathrm{ringing}(a,b)$ という呼出し中の状態に変化することを示している.

2.2　サービスルールの図式表記

ある時間におけるサービスの状態を表したものをシステム状態と呼ぶ. 本稿では，このようなシステム状態にサービスルールを適用した結果を分かり易く捉えるために，2.1 節の例 1，例 2 のようなサービスルールを図 1 のように図式表記する．枠で囲まれた部分はシステム状態，丸で囲まれた英文字はユーザを示す．また小文字はユーザ変数を表し，大文字は実際のユーザを表すユーザ値を表している．システム状態の間にある矢印はイベントを示す. ユーザ間にある矢印はその回線状況を表し，実線は通話中，点線は保留中を示す.

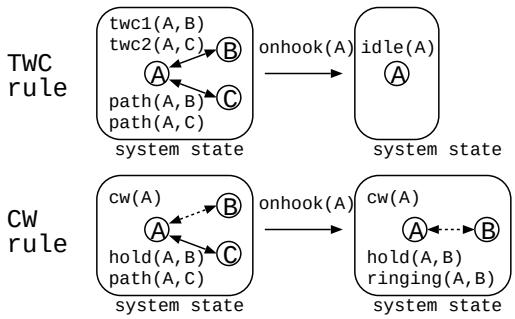

図 1: 図式表記の例

3　フィーチャインタラクション

通信ネットワークにおいて複数のサービスが並行に動作するとき，サービス間の競合によりシステム状態に矛盾が生じることがある．これをフィーチャインタラクションと呼ぶ．またフィーチャインタラクションはその特徴により 7 種類に分類することができる. 本稿ではその内で最も代表的なフィーチャインタラクションである異常状態の出現とデッドロックを動的に回避する動的回避法について述べる. デッドロックとはあるシステム状態においてあるユーザの実行可能イベントが無くなってしまいそのシステム状態から遷移できない状態を言う. また異常状態とは，イベント実行後のシステム状態において意味的に互いに矛盾する述語が含まれることを示す．例として，TWC 及

び CW が並行に動作しているユーザ A を想定し，その状況で起こり得る異常状態の出現を図 2に示す．

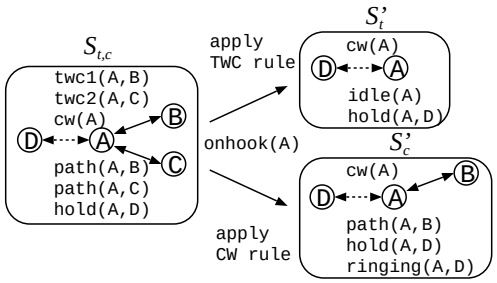

図 2: 出現例

図 2の $S_{t,c}$ は，図 1の 2 つの前条件を同時に満たすシステム状態である．システム状態 $S_{t,c}$ においてイベント onhook(A) が発生すると，TWC あるいは CW のルールが実行可能である．この時どちらのイベントを実行するかは一意には決定できず，フィーチャインタラクションの 1 つである非決定性ということになるのだが，本稿では一意に決定するものとし各場合ごとに論議する．まず TWC ルールを実行する場合，システム状態は S'_t に遷移する．S'_t において注目すべき述語は idle(A) かつ hold(A, D) である．これはユーザ A がアイドル状態でありながらユーザ D を一時的に保留にしていることを示す．故に，明らかに矛盾している．

同様に，CW ルールを実行する場合，システム状態は S'_c に遷移する．S'_c ではユーザ A がユーザ B と通話中であるにも関わらず，主音 (ベル) が鳴り呼出し中の状態であることを示す．故に，こちらも明らかに矛盾している．従って，S'_t 及び S'_c は異常状態である．

4 SA 状態回避法

本稿で議論する異常状態出現の動的回避法である SA 状態回避法 [3] の概要を以下に示す. SA 状態回避法では, 準異常状態と呼ばれる状態 (SemiAbnormal 状態と呼び以下 SA 状態とする) を回避することで異常状態に陥ることを回避している.

まずサービス h，r が並行に動作しているシステム状態 S_0 を考える．このとき，サービス h，r のイベントがランダムにそれぞれ p，q 回実行されたと仮定すると，S_0 は $n = p + q$ 回の遷移を経て S_n に到達する．ここで，S_n に到達するまでにシステム状態が異常状態にならないようにするために，システム状態は下記に示す対応規則を満たす必要がある (図 3参照).

以降では，システム状態において，サービス h あるいは r に関係する述語に限った状態をサービス h あるいは r に特化した状態という．

図 3では，横軸がサービス h に特化した状態系列 u，縦軸がサービス r に特化した状態系列 v を示している．システム状態の系列は二次元的に表され，サービス h のイベントの実行では横軸方向に遷移し (例えば，$S_{k_{p-1}} \rightarrow S_{i_p}$)，同様にサービス r のイベントの実行では縦軸方向に遷移する (例えば，$S_{i_p} \rightarrow \cdots \rightarrow S_n$)．そして，各 $S_x (0 \leq x \leq n)$ は両軸に対応している u と v を複合したシステム状態である (例えば，$S_n = u_p \cup v_q$).

対応規則とは，S_n においてサービス h のイベント e_{p+1} が実行されたとき，

$$S_{n+1} = u_{p+1} \cup v_q$$

が成立し，同様に，サービス r のイベント $\hat{e}_{q+1}$ が実行されたとき，

$$S_{n+1} = u_p \cup v_{q+1}$$

が成立することである．この対応規則が満たされていない状態は, 異常状態の可能性がありこの状態を SA 状態とする. SA 状態回避法では SA 状態と判定された場合そのイベントを実行しないことで異常状態を回避する方法である.

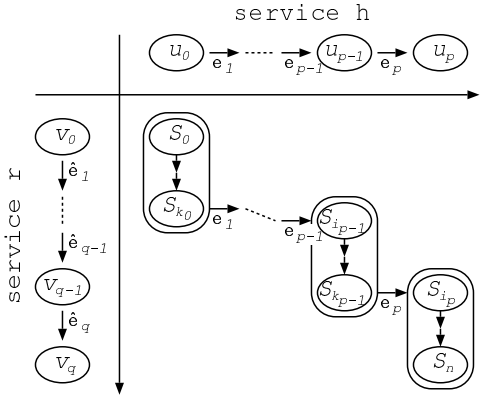

図 3: システム状態の対応

5 提案法

5.1 提案法

SA 状態回避法では SA 状態を回避することで異常状態を回避している. しかし, SA 状態への遷移を引き起こすイベントをユーザに中止させることは困難であり, また

ユーザのイベントを制限してしまうためユーザの実行可能イベントがなくなるデッドロックに陥る場合が確認されている．そこで本稿では SA 状態を回避するのではなく，サービスルールを必要に応じて追加することにより次状態を SA 状態から正常状態へと変換するという方法を提案する．

異常状態から正常状態への変換であるが，そもそも SA 状態のなかでも片方のサービスが終了しもう片方のサービスが正常に動作する状態は正常状態である．今，SA 状態が上記の状態であるときそのままで正常と判定し，そうでなければサービスルールを追加し上記の状態へと変更することにより SA 状態を回避する (図 4参照)．

提案法は大きく以下の 3 つのステップから成り立っている．

step1:イベントの実行によりサービスが終了したかの判定．終了したなら step2 へ．

step2:イベント修了後の状態が正常であるかの判定．正常であるなら無変更，正常でないなら step3 へ．

step3:矛盾を起こすサービスを終了させるサービスルールへと追加することでイベント実行後の状態を正常状態へと遷移させる．

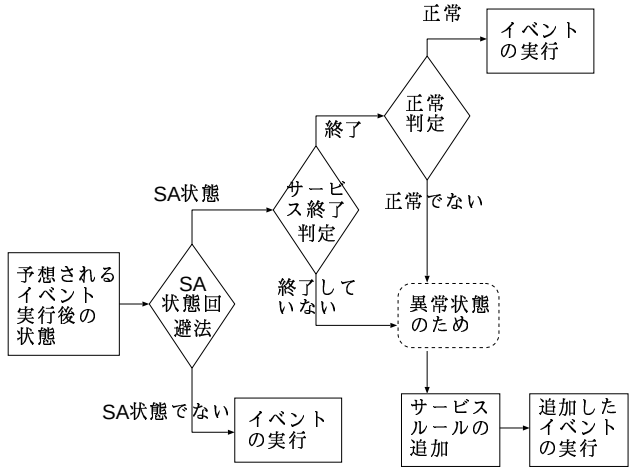

図 4: 提案法の流れ

また実際の判定方法としてサービスが終了したかの判定ではそのサービスに特化した状態 U_n が基本サービスのいずれかのルールの前条件と一致したとき終了したと判定する．ここで基本サービスとは全てのユーザが受けることのできるサービスでこれに CW や TWC などの付加サービスが付加されることで付加サービスの加入者が付加サービスを受けることができる．次に step2 のイベント修了後の状態が正常であるかの判定は片方のサービスが終了した時の状態 S_n が残りのサービスのいずれかのルールの前条件と一致したときそのサービスは正常に動作しているとする．最期に追加するサービスルールであるがこれは先に矛盾を起こすサービスを終了させるサービスルールと説明した．これは言い替えると矛盾を起すサービスに特化した述語を削除するサービスルールと言うことができる．具体的にはまずフィーチャインタラクションを起こしているサービスに加入しているユーザ A に関しては，矛盾を起こしているサービスのみに特化した述語 (他のサービスと共通でない述語) を削除すればよい．またそのサービスに関与している他のユーザに関しては，そのサービスに加入しているユーザ A がそのサービスを終了したということを伝える sig-onhook(A) を実行すればよい．さらにこのことはサービスに加入しているユーザ A に関する述語のみで考えた場合，前条件がイベント実行前のシステム状態で後条件が実行するイベントの後条件であるものと言うことができる．

5.2 正当性の証明

正当性の証明として以下の 3 つのことを図 5を用いて証明する．まず 1 つめとしてあるサービスに特化した部分が基本サービスの前条件と一致することでそのサービスが終了したと言えることを証明し，2 つめとして 1 を満たすときそのシステム状態がもう片方のサービスの前条件と一致することでそのシステム状態が正常状態であると言えることを証明し，最後に前条件が実行前のシステム状態で後条件は実行するイベントの後条件であるサービスルールへと追加することで異常状態から正常状態へと変更できることを証明する．また図 5において状態 H, R はそれぞれサービス H, R の状態を示しており，状態 S はサービス H, R の両方が動いている状態を示している．ここでイベント e_h により H は H' に S は S' へと遷移する．また A, C はそれぞれサービス H, R に特化した述語を示し，B はサービス H, R の共通の述語を示す．

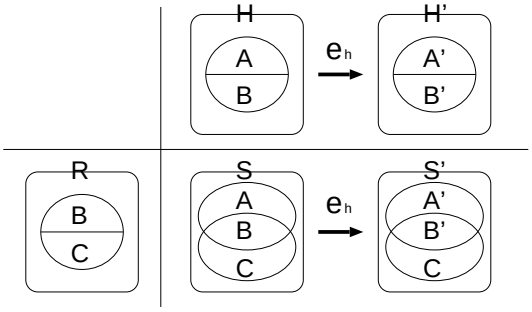

図 5: 対応表

1 : 今，$A' \cup B'$ が基本サービスの前条件と一致すると，

状態 H' は基本サービスが単独で動いているのと同じ状態であると言える．つまり付加サービスであるサービス H は付加されていない状態であると言え, 状態 H' においてサービス H は終了していると言える．またサービス H が終了しているということはサービス H に特化した部分である A' は空集合であると言える．

2：1 より $A' = \phi$ である．よって S'=$B' \cup C$ となる．ここで $B' \cup C$ がサービス R の前条件と一致するということは状態 S' はサービス R が単独で動いているのと同じ状態であると言える．ここでサービスは単独ではフィーチャインタラクションを起こさない．またサービスの前条件は正常であるので状態 S' は正常となる．

3：今, イベント e_h の代わりに $A, B, C[special]A', B'$ を実行するため S'=$A' \cup B'$ となる．ここで $A, B[e_h]A', B'$ より H'=$A' \cup B'$ は正常である．よって S'=$A' \cup B'$ は正常であると言える．

5.3 適用例

5.3.1 適用例 1

今, 例として状態 S_{00} を定義する．ここで TWC サービスのイベント $Flash(C)_{TWC}$ が実行された時の次状態 S_{01} は SA 状態回避法により準異常状態と判定されるため提案法を適用していく (図 6参照)．まず step1 としてサービスが終了したかの判定であるが, TWC サービスに特化した状態 U_1 は基本サービスの前条件と一致するため U_1 においては TWC サービスは終了していると判定される．よって次に stpe2 のその状態が正常であるかどうかを判定する．これは TWC サービスが終了している状態 S_{01} においてもう片方のサービス CW の前条件と一致するため正常であると判定される．
よってこの場合は SA 状態回避法により SA 状態と判定されるが提案法により正常状態であることが分かる．

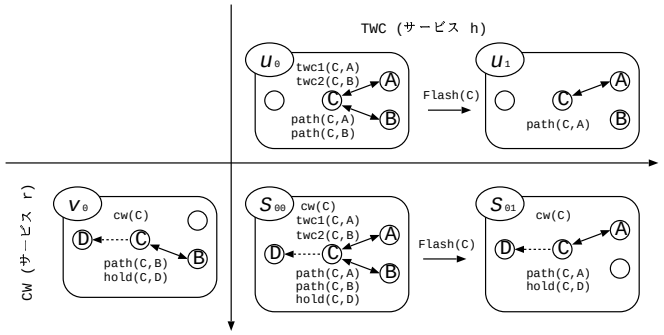

図 6: 適用例 1

5.3.2 適用例 2

次に例 1 と同じ状態 S_{00} において CW サービスのイベント $Flash(C)_{CW}$ が実行された場合を考える．この場合も例 1 と同様に次状態 S_{10} は SA 状態であると判定されるため提案法を適用する (図 7参照)．まず step1 としてサービスが終了したかの判定であるが, 状態 V_1 は基本サービスのどの前条件とも一致しないため CW サービスは終了していないと判定される．よって提案法の step3 に従いサービスルールを追加することで状態 V_1 を SA 状態から正常状態へと遷移させる．追加するサービスルールは前条件として状態 S_{00} を持ち後条件として状態 V_1 を持つ．つまり以下のイベントへと追加する．

$$\text{twc1}(C,A),\text{twc2}(C,B),\text{cw(C)},\text{path}(C,A),\text{path}(C,B),$$
$$\text{hold}(C,D)\ [\text{event}(a)]\ \text{cw(C)},\text{path}(C,D),\text{hold}(C,B)$$

サービスルールを追加することにより次状態は S_{01} ではなく V_1 へと遷移する．この状態はユーザ C がユーザ D と通話中でありユーザ B を保留している状態である．またこの状態は CW サービスが単独で動いている状態であり, 正常状態である．

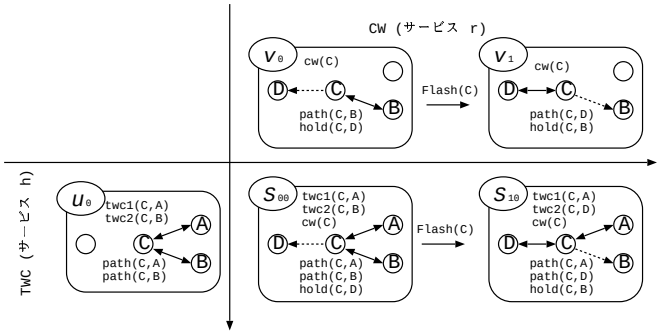

図 7: 適用例 2

6 シミュレーション実験

6.1 実験の目的

提案法の有効性を確認するために，シミュレータを作成し実験を行った．実験の目的としては以下の 2 つが挙げられる．

(1) デッドロックと異常状態は回避できているか
(2)SA 状態の中の正常状態の割合はどの程度か

1 つめのデッドロックと異常状態については，初期状態から到達可能な全ての状態においてデッドロックと異常状態が検出されたかを調べる．これにより，提案法がデッ

ドロックと異常状態の回避に効果があるかを調べる．また２つめの SA 状態の中の正常状態の割合を調べることで，SA 状態回避法で回避されていた SA 状態の中から提案法によりどの程度の状態が救済されているかを調べる．

ここで，実験はシミュレータにある初期状態を与え，その状態から到達可能な全ての状態についてその状態がどのような状態であるかを調べることで行っていく．またシミュレータに与えた初期状態は以下のものである．まず初めに，フィーチャインタラクションを起こすサービスとして，CW サービスと TWC サービスを用いた．これは，SA 状態回避法を適用した実験において，SA 状態とデッドロックが数多く検出されたサービスの組み合せであるためこれらのサービスを撰択した．またシミュレータに投入した初期状態であるがユーザ数は 4 人とし，各ユーザがアイドル状態であるとした．

6.2 結果と考察

1 つめの目的であるデッドロックと異常状態の数を調べた結果を表 1 に示す．

表 1: デッドロックと異常状態の数

	デッドロックの数	異常状態の数
SA 状態回避法のみ使用	54	0
提案法を使用	0	0

表 1 より SA 状態回避法のみを適用した場合異常状態は回避できているがデッドロックに陥っていることが分かる．さらに SA 状態のみでなく提案法を適用することで，異常状態とデッドロックを回避できていることが分かる．この結果より，提案法は異常状態とデッドロックの回避に効果があることが分かる．

次に SA 状態の中の正常状態の割合について表 2 に示す．

表2： SA状態の中の正常状態の割合

SA状態でないと判定	SA状態と判定	
20773	5402	
	サービス終了でない	サービス終了により正常状態
	906	4496
	サービスルールの置換	

表 2 より SA 状態でないと判定された状態が 20773 であるのに対し，SA 状態と判定された状態は 5402 であった．これは SA 状態回避法では到達可能な状態の内，約 1/4 の状態を回避していたことになる．しかし，提案法を使用することで 5402 の SA 状態の内，4496 の状態を正常状態と判定することができた．よって提案法は SA 状態の中の正常状態の判定にも効果があることが確認できた．また提案法によってもサービス終了でなく正常でないと判定された 906 の状態についてはサービスルールの追加により正常状態へと遷移できていることが確認された．
以上の結果より，提案法は異常状態とデッドロックを動的に回避することができ，SA 状態の中の正常状態の判定にも効果があることが確認された．

7 まとめ

本稿では, 異常状態の出現によるフィーチャインタラクションの動的回避法である SA 状態回避法を改善することでデッドロックと異常状態を回避できる動的解消法を提案し, その正当性を証明した. 提案法は SA 状態にサービスルールを追加することで SA 状態を正常状態へと遷移させる手法である. またシミュレータを作成し実験を行うことでその有効性を確認した.
今後の課題として, 実験による有用性の確認や適用時における通信オーバヘッドを実験的に評価することや，2 つのサービス間のみでなく 3 つ以上のサービスにより起こるフィーチャインタラクションについての有効性の確認などがある.

参考文献

[1] Dirk O.Keck, Paul J.Kuehn : “The feature and service interaction problem in telecommunications systems : A survey”, IEEE Trans. Software Engineering, 24, 10, pp.779-796, October 1998.

[2] Y.Kakuda, A.Inoue, H.Asada, T.Kikuno, T.Ohta: “A dynamic resolution method for feature interactions and its evaluation”, Proc. FIW’95 pp.97-114,1995.

[3] 栗栖陽介, 井上伸二, 角田良明:“異常状態の出現によるフィーチャインタラクションの動的解消法”, 信学技報 IN99-34,July 1999.