ソフトウェアテストシンポジウム 2005
JaSST'05: Japan Symposium on Software Testing 2005

抽象化に基づくリモートコンポーネントベースシステムの複雑度測定

中川 知基† 鷲崎 弘宜‡ 斉藤 勇樹† 深澤 良彰†

†早稲田大学大学院理工学研究科〒169-8555 東京都新宿区大久保 3-4-1
‡国立情報学研究所 〒101-8430 東京都千代田区一ツ橋 2-1-2
E-mail: †{20wins, g-person, fukazawa}@fuka.info.waseda.ac.jp, ‡washizaki@nii.ac.jp

あらまし リモートコンポーネントベースシステムの長期にわたる運用を考えるとき, システムは高い保守性を備える必要がある. 保守性の判定法として複雑度の測定法があるが, 従来の測定法はオブジェクト指向クラスの結合関係から複雑度を測定するため, コンポーネントレベルでの複雑さを反映していない. 本稿では, リモートコンポーネントの特徴を考慮してシステムを抽象化し, 得られる構造に対して構成要素間の結合関係に基づいて複雑度を測定する手法を提案する. EJB アプリケーションについて実験を行った結果, 提案する測定法が機能拡張後の保守作業において有効であったため、保守において役立つ指標となる可能性がある.

キーワード リモートコンポーネントアーキテクチャ, 保守性, 複雑度, ソフトウェア抽象化, EJB

Measuring Complexity of Remote Component-based System by Abstraction

Tomoki Nakagawa† Hironori Washizaki‡ Yuhki Saito† and Yoshiaki Fukazawa†

†Science of Engineering, Waseda University 3-4-1 Okubo, Shinjuku-ku, Tokyo, 169-8555 Japan
‡National Institute of Informatics 2-1-2 Hitotsubashi, Chiyoda-ku, Tokyo, 101-8430 Japan
E-mail: †{20wins, g-person, fukazawa}@fuka.info.waseda.ac.jp, ‡washizaki@nii.ac.jp

Abstract A system needs to be equipped with the ease of carrying out of high maintenance when long-term employment of a remote component-base system is considered. Although there is a measuring method of the degree of complexity as a diagnosis of the ease of carrying out of maintenance, since the conventional measuring method has measured the degree of complexity from the relation of an object-oriented class, it is not reflecting the complexity in a component level. In this paper, the technique of measuring the degree of complexity is proposed in consideration of the feature of a component-base system to the structure which abstracted the system appropriately. As a result of experimenting about EJB application, it checked that the degree of complexity obtained reflected conservativeness appropriately.

Keyword Remote component architecture, maintainability, complexity, software abstraction, EJB

1. はじめに

Enterprise JavaBeans(EJB)に代表されるリモートコンポーネントベースシステム（Component-base System; 以下, CBS）は, 長期の運用を実現するために高い保守性を兼ね備える必要がある. システムの保守性を評価する方法として, 複雑度の測定が挙げられる.

CBS を構成する各コンポーネントは, 主にオブジェクト指向プログラミング言語によって実装されるため, CBS の複雑度測定に, オブジェクト指向に基づく複雑度測定法を利用できる. しかしながら, 従来のオブジェクト指向複雑度測定法は, 測定手順に抽象化手順を組み込まず, オブジェクト指向クラスとクラス間の結合関係をそのまま測定対象とするため, 実装に用いるフレームワークやプログラミング言語の影響を大きく受け, システム全体の本質的な複雑度を反映できない可能性がある. 実装工程より以前の上位工程の成果物である分析／設計モデルに従来の測定法を適用することも可能であるが, 実装工程におけるソースコードと上位工程における分析／設計モデルが適切に対応付けられていない可能性があり, 測定結果は信頼性を欠く.

本稿では, コンポーネントアーキテクチャとして EJB を対象とし, ソースコードのみから得られる状況

において，リモートコンポーネントの特徴を考慮して CBS の実装クラス集合を段階的に抽象化し，得られる構造の複雑度を測定する手法を提案する．

2. リモートコンポーネントと EJB

コンポーネントベース開発では，まず，基盤となるソフトウェアアーキテクチャであるコンポーネントアーキテクチャを決定し、コンポーネントアーキテクチャの規格に従って動作可能なソフトウェア部品としてのコンポーネントを再利用，もしくは新規に開発し，コンポーネント集合を組み合わせることによって，新しい高品質・大規模・多様なソフトウェアを迅速に開発する．

2.1.コンポーネントアーキテクチャ

コンポーネントアーキテクチャは，コンポーネントの利用方法の違いから，ローカルコンポーネントアーキテクチャとリモートコンポーネントアーキテクチャの 2 種に大別できる[1].

ローカルコンポーネントアーキテクチャとは，コンポーネントの機能を利用する側とコンポーネントが，同一環境上に存在するものをさす．具体的なローカルコンポーネントアーキテクチャには，OLE/COM, JavaBeans, IntelligentPad などがある．

一方，リモートコンポーネントアーキテクチャとは，コンポーネントの機能を利用する側とコンポーネントが，異なる環境上に存在するものをさす．具体的なアーキテクチャとして，DCOM/ActiveX，Enterprise JavaBeans(EJB), CORBA Component Model (CCM)/CIDL などがある．

これまでに，ローカルコンポーネントの品質測定法がいくつか提案されているが[2]，リモートコンポーネントの特徴を考慮したシステム複雑度は提案されていない．

2.2.EJB

本稿では，EJB を実装コンポーネントアーキテクチャとして，リモートコンポーネントを組み合わせて得られる CBS 全体の複雑度を測定し，システムの保守作業を円滑に支援することを目指す．

EJB は，ビジネスエンティティとビジネスロジックをカプセル化するためのリモートコンポーネントアーキテクチャであり，Java 言語で記述される[3]．EJB におけるコンポーネントをエンタープライズ Bean と呼ぶ．エンタープライズ Bean は，コンポーネントインターフェース（エンタープライズ Bean の機能を外部に対して公開するためのインターフェース），ホームインターフェース（エンタープライズ Bean の生成管理を決定するインターフェース），および，1 つのエンタープライズ Bean 実装クラス

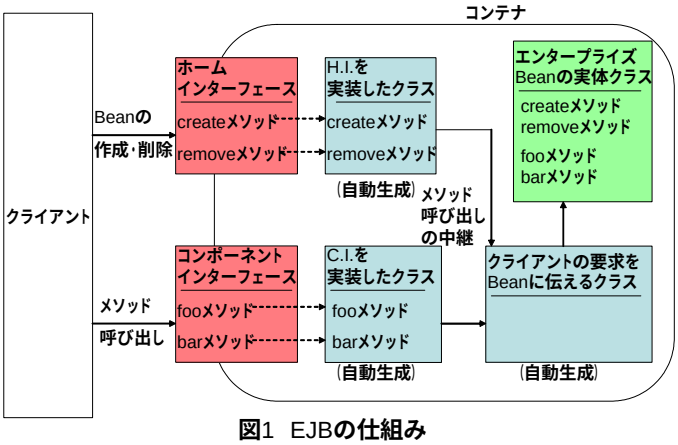

図1 EJBの仕組み

（javax.ejb.EnterpriseBean を間接または直接に実装したクラス）から構成され，個々のエンタープライズ Bean を EJB コンテナ内で組み合わせて使用する．図 1 は個々の EJB の仕組みを表している．コンテナにエンタープライズ Bean が配備されると，コンテナは 2 つのインターフェースを実装したクラス，エンタープライズ Bean 実装クラスに要求を伝えるクラスを自動生成する．クライアントは，必ず 2 つのインターフェースを介してエンタープライズ Bean を生成／削除し，メソッド呼び出しを行う．

3. オブジェクト指向複雑度測定法

オブジェクト指向システムの複雑度を測定するための測定法として，結合度測定法がある．結合度測定法は，システムを構成するクラス間の関係に着目し，結合度や凝集度を算出する測定法であり，オブジェクト指向を最も象徴した測定法である[4]．代表的な結合度測定法として，COF(Coupling Factor)[5]がある．COF の定義は次式で与えられる．

$$COF(S)=\begin{cases}\dfrac{\sum_{i=1}^{|S|}\sum_{j=1}^{|S|} isClient(c_i,c_j)}{|S|^2-|S|-(2\sum_{k=1}^{|S|}|Descendents(c_k)|)} & (|S|\neq 0,1)\\ 0 & (|S|=0,1)\end{cases}$$

S={c_1, … , $c_{|S|}$}は，測定対象となるシステムを表す．c は，システムを構成するクラスを表す．Descendents(c) は，クラス c のサブクラス数を表す．isClient(c_1,c_2)は，c_1 と c_2 が継承関係になく，$c_1 \neq c_2$ であり，かつ，c_1 が c_2 の属性やメソッドを参照している場合には，1 を表し，それ以外のときは，0 を表す式である．例えば図 2 のオブジェクト指向クラスの構造に対して COF を適用すると，以下の値が得られる．

$$COF(\{c_{1,\ldots,}c_{28}\})=\frac{36}{28^2-28}\quad 0.048$$

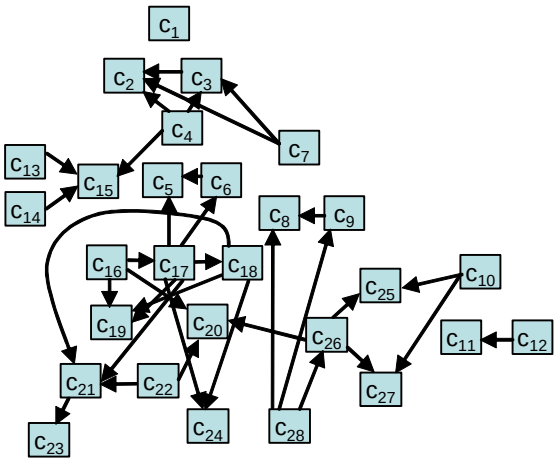

図2　オブジェクト指向クラスの構成

COF は，測定手順に抽象化操作を組み込まず，システムを構成するオブジェクト指向クラスとクラス間の結合関係をそのまま測定対象とするため，実装に用いるアーキテクチャおよびフレームワークの影響を大きく受ける．例えば，エンタープライズ Bean は主に，2 種の Java インターフェース（ホームインターフェースとコンポーネントインターフェース）と，対応する 1 つの実装クラスを作成することで実現される．このとき，2 種のインターフェースを継承するクラスはコンテナ（もしくは EJB 配備ツール）によって自動生成されるため，対応する実装クラスとインターフェース間にはプログラミングレベルにおいて関連が存在しない．従って，単純に COF を CBS に適用する場合に，得られる測定値はシステムの本質的な複雑度を反映できない可能性がある．また，COF は，すべてのクラスを等価のもとして計測しているため，CBS においてコンポーネントを単位とした保守作業の容易さを反映しない可能性がある．

4. コンポーネントベース複雑度測定法

COF の仕組みを発展させて，リモートコンポーネントの特徴を考慮したコンポーネントベース複雑度測定法 CCOF(Component Coupling Factor)を提案する．CCOF は，対象とする CBS のオブジェクト指向クラス集合を抽象化して得られる抽象化クラス集合 $S=\{B_1, \ldots, B_{|S|}\}$ を測定対象とする．CCOF の定義を以下に示す．

$$CCOF(S)=\begin{cases}\dfrac{\sum_{i=1}^{|S|}\sum_{j=1}^{|S|} hasReferences(B_i,B_j)}{|S|^2-|S|} & (|S|\neq 0,1)\\ 0 & (|S|=0,1)\end{cases}$$

S は測定対象となるシステムを表す．B は，1 つのエンタープライズ Bean をまとめた抽象化クラス，もしくは，システムにおいてコンポーネント（エンタープライズ Bean）と同程度の機能規模を持った抽象化クラスを表す．hasReferences(B_1,B_2)は，$B_1 \neq B_2$，かつ，B_1 が B_2 を参照している場合に 1 を，それ以外の場合は，0 を表す式である．CCOF の式の分母は，値を 0～1 に正規化するための補正項である．

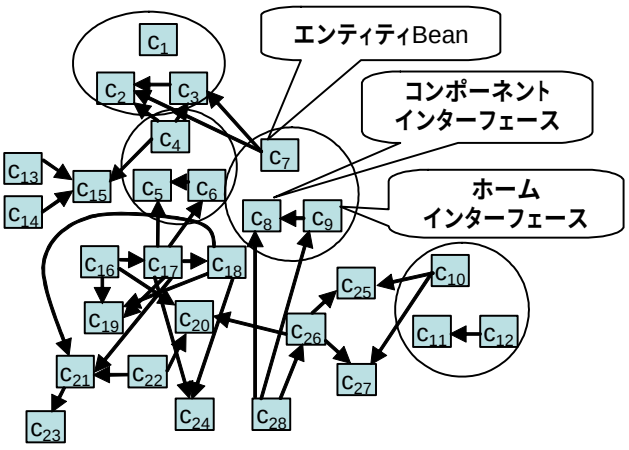

図3　コンポーネントの特定

4.1. システムの抽象化

システムの抽象化は，以下の 3 つの手順から成る．

1. システムのクラス関連グラフの作成
2. コンポーネントの特定
3. コンポーネント以外のクラス部分集合の抽象化

以降において，各手順について詳しく述べる．

(1) クラス関連グラフの作成

対象とする CBS のクラス集合のクラス関連グラフ(CRG)を作成する．CRG とは，クラスおよびインターフェースに対応する頂点の集合 V と，クラス・インターフェース間の参照／継承関係を表す有向辺（図示：→）の集合 E から構成される有向グラフである[6]．参照関係として，他方のクラス・インターフェースのフィールド参照，メソッド参照，および，クラス・インターフェースの型指定を扱う．例えば，図 2 のようなクラス関連グラフを作成する．以降において，このグラフについて抽象化手法を適用する．

(2) コンポーネントの特定

対象とする CBS のクラス集合において，コンポーネントを特定し，コンポーネントを構成するクラスおよびインターフェースを 1 つの抽象化クラスにまとめる．本稿では，コンポーネントアーキテクチャとして EJB を対象とするため，本手順においてエンタープライズ Bean を特定する．

具体的には，エンタープライズ Bean の実装クラス，コンポーネントインターフェース，ホームインターフェースの 3 つ組を，エンタープライズ Bean 配備記述子およびクラス・インターフェース名の対応関係から特定し，1 つの抽象化クラスにまとめる．コンポーネ

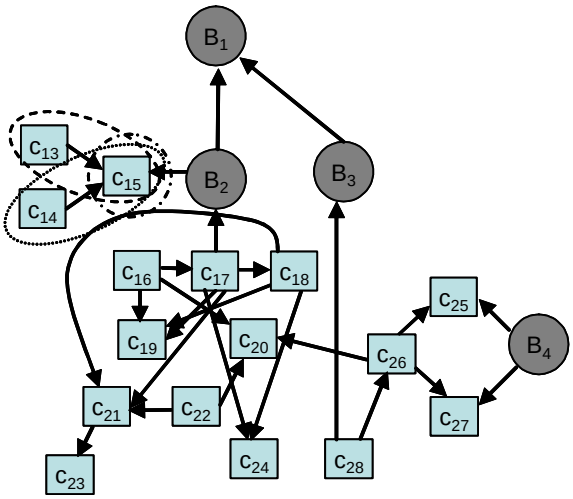

図4　　抽象化手順(a)

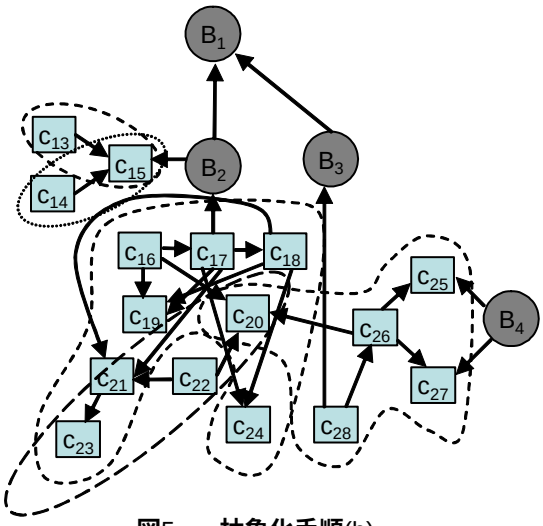

図5　　抽象化手順(b)

ントを構成するクラス・インターフェースへの外部からの参照は，抽象化後の抽象化クラスへの参照へ対応付ける．例として，クラス集合からエンティティ Bean を特定して抽象化クラスを作成する様子を図 3 に示す．

例えば，c_7 はエンタープライズ Bean の実装クラスとしてのエンティティ Bean, c_8 はコンポーネントインターフェース，c_9 はホームインターフェースである．このとき，これらの 3 つをまとめて抽象化クラス B_3 とする．同様に，c_1～c_3，c_4～c_6，c_{10}～c_{12} もそれぞれ B_1，B_2，B_4 にまとめる．この抽象化手順の結果，図 4 のようなグラフが得られる．

(3) クラス部分集合の抽象化

コンポーネントの特定手順におけるコンポーネント以外のクラス集合は，参照／継承到達可能なクラス部分集合を 1 つの抽象化クラスに対応付けることにより抽象化する．本手順は，以下の手順(a)～(c)より得る．

手順(a)：抽象化されていないクラス・インターフェースについて，クラス関連グラフ上で参照／継承到達可能な抽象化されていないクラス集合

$$R(c)=\{c_r \mid c \overset{*}{\rightarrow} c_r \wedge (c_r \text{は抽象化クラスではない})\}$$

を求める．ここで，任意の *CRG* $\Gamma=(V, E)$において，任意の頂点 u から v へ 1 つ以上の有向辺を経由して到達できるか，または u=v の時，u から v へ参照／継承到達可能であると呼び，$u \overset{*}{\rightarrow} v$ と表す．

例えば図 4 は，前の手順においてエンタープライズ Bean に対応して抽象化された抽象化クラス B_1～B_4 と，抽象化されていないクラス c_{13}～c_{28} から構成されるシステムを表す．ここで，クラス c_{13}～c_{15} についてそれぞれ $R(c_{13})=\{c_{13},c_{15}\}$, $R(c_{14})=\{c_{14},c_{15}\}$, $R(c_{15})=\{c_{15}\}$を得る．

手順(b)：参照到達可能なクラス集合をすべて仮に抽象化クラスにまとめた後に，参照／継承到達可能なクラス集合の包含関係について他の集合を含む側の抽象化クラスを残す．例えば図 5 において，クラス c_{13}～c_{15} に関する参照／継承到達可能なクラス集合の包含関係は，

$$R(c_{13}) \supset R(c_{15}),\ R(c_{14}) \supset R(c_{15})$$

である．従って，$R(c_{13})$および $R(c_{14})$をそれぞれ異なる抽象化クラスにまとめることができる．同様にしてすべてのクラスの参照／継承到達可能なクラス集合を調査すると，図 5 のシステムにおけるエンタープライズ Bean に対応した抽象化クラス以外のクラス集合は，

$$R(c_{13}),\ R(c_{14}),\ R(c_{16}),\ R(c_{22}),\ R(c_{28})$$

の 5 つのクラス部分集合によって網羅できる．それぞれを抽象化クラス B_5, B_6, B_7, B_8, B_9 にまとめる．

手順(c)：抽象化クラス間の関連付けを行う．具体的には，抽象化クラス B について対応する抽象化する前のクラス集合を $A^{-1}(B)$ と表すとき，抽象化クラス B_i と B_j について，抽象化前のクラス集合 $A^{-1}(B_i)$, $A^{-1}(B_j)$中に，何らかのクラス・インターフェースが共通に含まれるとき，B_i と B_j 間の双方向に参照関係を与える．また，$A^{-1}(B_i)$中のあるクラス・インターフェースから，$A^{-1}(B_j)$中のあるクラス・インターフェースに直接の参照関係があるときに，B_i から B_j への参照関係を与える．

例えば図 5 において，$A^{-1}(B_5)$と $A^{-1}(B_6)$は共通にクラス c_{15} を含む．従って，B_5 と B_6 の間に双方向の参照関係を与える．このようにして，図 5 に示される抽象化前のシステムは，図 6 のように抽象化される．

最後に，得られる抽象化後の抽象化クラス集合について，CCOF を適用し複雑度を得る．例えば図 6 の場合は以下の値を得る．

$$CCOF(\{B_1,...,B_9\}) = \frac{15}{9^2-9} \\ 0.208$$

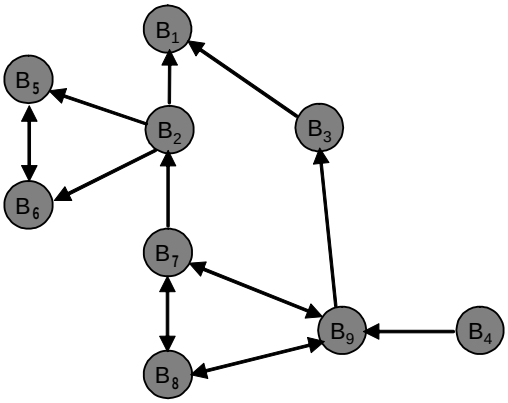

図6　抽象化手順(c)

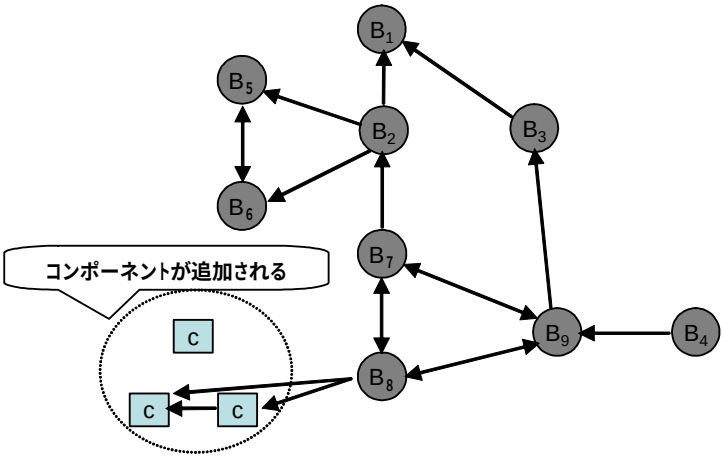

図7　エンタープライズBeanの追加

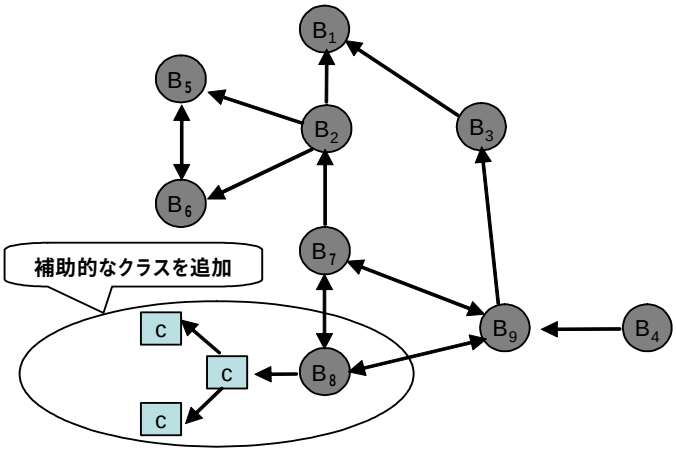

図8　コンポーネント以外のクラスを追加

4.2. 機能追加による保守作業の考察

システムが改変された後の保守作業の容易さについて，抽象化により得られたグラフと抽象化前のグラフとを比較する．例えば，図 7 に示すように，新しい機能を加えるためにエンタープライズ Bean を図 6（図 2）のシステムに追加する．このシステム S’とする．

このとき, このシステムに対して COF, CCOF を求めると以下の値を得る．

$$COF(S') = \frac{39}{31^2 - 31} \quad 0.043$$

$$CCOF\ (S') = \frac{16}{10^2 - 10} \quad 0.178$$

一方，補助的な機能を追加するために，図 8 に示すように，コンポーネント以外のクラスを図 6（図 2）のコンポーネント以外の抽象化クラスに追加する．このシステム S’’とする．このとき，COF, CCOF の値として以下を得る．

$$COF(S'') = \frac{39}{31^2 - 31} \quad 0.043$$

$$CCOF(S'') = \frac{15}{9^2 - 9} \quad 0.208$$

ここで，S’は，新たな再利用／交換可能な安定的なエンタープライズ Bean を追加することにより、補助的な機能を追加した S’’よりも保守作業が容易となることが予想される．

そこで S’と S’’の COF，CCOF の両測定値を比較すると，値の大小について以下の関係を得た．

COF(S’) = COF(S’’)

CCOF(S’) < CCOF(S’’)

COF では, S’と比較して S’’において複雑度が変わらないのに対し，CCOF では，複雑度が大きくなった．このことから，拡張によるシステム変更後の保守の容易さをコンポーネントレベルで見たとき，CCOF は COF と比較して保守性をより適切に反映していると考えられる。

5. 測定実験

実際に用いられている Web 上・書籍上の 10 個の EJB アプリケーション(S_1～S_{10})を実験サンプルとして，COF および CCOF をそれぞれ適用し，得られる両測定法の測定値の比較を行った．各サンプルのプロジェクト名, 元のクラス・インターフェース数, および, CCOF における抽象化後の抽象化クラス数を表 1 に示す．また，各サンプルの COF および CCOF の測定値を図 9 に示す。

両測定値の大きい順にサンプルを並べたときに，COF の値の大小関係と CCOF の値の大小関係が逆転しているパターンは，次の 13 通りであった．

(S_1,S_9), (S_1,S_{10}),(S_3,S_4), (S_3,S_5), (S_3,S_6),
(S_4,S_5), (S_4,S_7), (S_5,S_6), (S_5,S_7), (S_5,S_{10}),
(S_6,S_7), (S_7,S_{10}), (S_9,S_{10}).

ここで，(S_i,S_j)とは，

$(COF(S_i)<COF(S_j) \wedge CCOF(S_i)>CCOF(S_j)) \vee$
$(COF(S_i)> COF(S_j) \wedge CCOF(S_i)<CCOF(S_j))$

を満たすサンプル S_i, S_j の対を表す．これらの各パターンでは，両サンプルを比較した場合，実装クラスレベルで見たシステムの複雑さの大小関係と，コンポー

ID	プロジェクト名	抽象化前クラス数	抽象化後クラス数
S_1	Financial Brokerage Service	72	11
S_2	Financial System Analysis	72	39
S_3	ITracker	198	25
S_4	Roomba	82	17
S_5	Virtual Shopping Mall	172	23
S_6	WebStore	89	10
S_7	xPetStore	110	13
S_8	PetStore	326	37
S_9	Book	52	11
S_{10}	C3J	45	12

表 1　サンプルのクラス数と抽象化クラス数

ネントレベルで見たシステムの複雑さの大小関係は一致していない.

例えば，$COF(S_1) < COF(S_9)$ より，S_1 と比較して S_9 はクラスレベルで見た場合により複雑であり，システムの保守作業がより困難と考えられる.しかしながら，コンポーネントレベルでの複雑度は $CCOF(S_1) > CCOF(S_9)$ であるため，S_9 は S_1 と比較してコンポーネント間の結合関係がより疎であり，コンポーネント内部で完結するクラス間の結合関係をより多くもつ（すなわちコンポーネント内部における凝集関係が密である）ことを示す．人手によるレビューの結果，S_9 のほうが，S_1 に比べ疎結合・高凝集な構造を持つシステムであり，CCOF はコンポーネントをシステムの構成単位とした場合の保守作業の容易さを適切に表していることがわかった.

なお，得られた結果に対する区間推定[7]の結果は，COF が 0.0217～0.0650, CCOF が 0.0589～0.124 であった．今後，各サンプルの EJB アプリケーションとしての設計の妥当性を検証し，得られた区間が，EJB アプリケーションとして適切な設計が行われていることを表すかどうかを確認する予定である.

6.CCOF の性質の検証

CCOFの測定法としての論理的正当性を示すために，数学的な性質について考察する．Briandらが提案する BMB Framework [10]は，結合度に基づく測定法が正当かどうかを示す必要条件である.BMB Frameworkでは，次の6つの条件を満たすことを示せばよい.

- 正規性

結合度は，0～1で正規化される.

$$0 \leqq CCOF(S) \leqq 1$$

（証明）

CCOF値が正規母集団からの標本であるかどうかを検証する.「データの分布は正規分布である」と言う帰無仮説のもとで検定を進める．ここでは，危険率を5%とする.

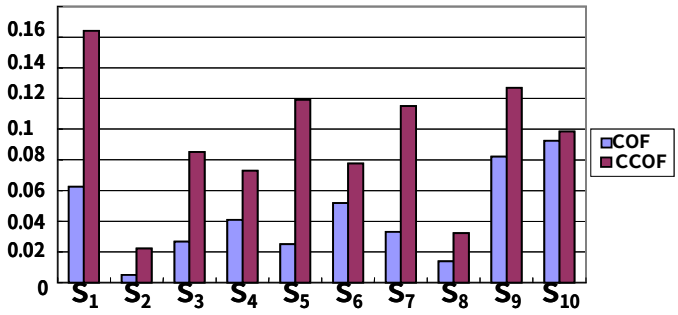

図9　　COF・CCOFの測定値

CCOFについて，χ^2値，$\chi^2(0.95)$値，P値（上側確立）を求めると，以下のようになる.

測定法	χ^2値	P値（上側確立）	$\chi^2(0.95)$
CCOF	0.475	0.490	3.84

χ^2値 $< \chi^2(0.95)$値であるので，棄却域には入らない.従って,「データの分布は正規分布とみなせる」という帰無仮説は棄却されない．P値についてみると0.342より，危険率5%では帰無仮説は棄却されない．従って，CCOFの測定値の分布は，正規分布に従う.

- 非負性

あるシステムの結合度は，非負である.

（証明）

CCOFの定義式より，

(i)$|S| = 0, 1$のとき$CCOF(S) = 0$

(ii)$|S| > 1$のとき$hasReference(B_i, B_j) = 0, 1 \geqq 0$かつ $|S|^2 - |S| > 0$より$CCOF(S) \geqq 0$

(i),(ii)より，CCOFの非負性が証明された.

- ヌル値

あるシステムの結合度は,そのシステム中に互いに関係するコンポーネントがないとき，結合度は0である.

（証明）

(i)$|S| = 0, 1$のとき定義式より，$CCOF(S) = 0$

(ii)$|S| > 1$: システム中に互いに関係するコンポーネントがないとき，定義式より，$hasReference(B_i, B_j) = 0$ このとき，定義式は，$CCOF(S) = 0$

(i),(ii)より，関係がないシステムのCCOF値は0である.

- 単調性

あるシステムSに新しい関係を追加したシステムをS'とすると，S'の結合度はSの結合度より大きい.

（証明）

(i)$|S| = 0, 1$のとき: 新しい関係を追加することは不可能.また,自己ループを持つこともできない.よって,

$$CCOF(S) = CCOF(S')$$

である．
(ii)|S| > 1のとき: 新しい関係を追加したとき
hasReference(Bi, Bj)の値は増加し，|S|の値は不変．よって，CCOF(S) < CCOF(S’)
(i),(ii)より，CCOFの単調性が証明された．

- クラスの併合

あるシステムS中の2つのコンポーネントB_1とB_2をB’で置換したシステムをS’とすると

$$CCOF(S) \geqq CCOF(S')$$

が成立する．
（証明）
(i)|S| = 0, 1のとき: コンポーネントの併合は, 不可能．よって，CCOF(S) = CCOF(S’)
(ii)|S| > 1のとき: コンポーネントの併合により，コンポーネントの総数は減少するので，|S| > |S’|
また，B1, B2の併合により，そのコンポーネント間の参照関係は少なくとも１つ失われるので，システムSの参照総数を|R|とし，S’の参照総数を|R’|とすると，

$$|R| > |R'| \geqq R - 1$$

今，CCOFの値が0～1で正規化されているので，

$$1 \geqq CCOF(S) = |R| / (|S|^2 - |S|) \geqq 0$$

が成り立つ．そこで，以下が成り立つ．

$$\begin{aligned} &CCOF(S) - CCOF(S') \\ &= \{|R| / (|S|^2 - |S|)\} - \{|R'| / (|S'|^2 - |S'|)\} \\ &\geqq \{|R| / (|S|^2 - |S|)\} - \{|R - 1| / (|S|^2 - |S| - 1)\} \\ &= \{|R|(|S|^2 - |S| - 1) - (|R| - 1)(|S|^2 - |S|)\} / \\ &\quad (|S|^2 - |S|)(|S|^2 - |S| - 1) \\ &= (- |R| + |S|^2 - |S|) / (|S|^2 - |S|)(|S|^2 - |S| - 1) \end{aligned}$$

条件より，|S| > 1すなわち，|S|は2以上の整数であるので，$(|S|^2 - |S|)(|S|^2 - |S| - 1) > 0$ を満たすので，分子のみを考える．

$$1 \geqq CCOF(S) = |R| / (|S|^2 - |S|) \geqq 0$$

より，$|R| \geqq |S|^2 - |S| \Leftrightarrow - |R| + |S|^2 - |S| \geqq 0$
であるので，以下が成り立つ．

$$CCOF(S) - CCOF(S') \geqq 0$$

(i),(ii)より，CCOF(S)≧CCOF(S’)が証明された．

- 無関連クラスの併合

あるシステム C 中の関連を持たない2つのクラス c_1 と c_2 を一つにしたクラスを c’とすると，

$$CCOF(S) = CCOF(S')$$

が成り立つ．
（証明）
これは，COF, CCOFともに満たさない．

以上より，CCOFは結合度測定法の必要条件の大部分を満たすことがわかった．

7.CCOF の独立性検証

CCOF が，COF に対して独立しており，新たな測定法として冗長ではなく有効であることを検証する．検証法には, 主成分分析[8]および相関分析[7]を適用した．分析の手順を以下に示す．

- 手順 1: 各サンプルに対する各測定法による測定値を成分としたベクトルを求める．
- 手順 2: 各ベクトルに対し，各成分の相関係数を算出し，相関行列 R を求める．
- 手順 3: 相関行列 R の固有値を算出する．ここでは, 各固有値をλ_1, λ_2とする．ただし，$\lambda_1 > \lambda_2$ であり，λ_j に対応する固有ベクトルは第 j 主成分と呼ばれている．
- 手順 4: 各主成分の累積寄与率α_1, α_2を求める．

$$\alpha = \frac{\sum_{t=1}^{j} \lambda_t}{\sum_{t=1}^{n} \lambda_t}$$

先の 10 個のサンプルについて，COF および CCOF の測定値について主成分分析を施し，累積寄与率を算出すると，第 1 主成分の累積寄与率が 0.805，第 2 主成分の累積寄与率が 1.000 であった．この結果は，第 1 主成分の累積寄与率が 0.85 未満であることから，使用したサンプルの量測定値は，実質 2 次元の広がりを持っていることを示す．従って，CCOF は COF に対して冗長ではないことがわかる．また，両測定法の測定値間の相関係数は 0.61 であった. 強い相関は確認できないため，CCOF は COF に対して冗長ではなく，独立していることがわかる．

8. 関連研究

Egyed は，UML で表されるオブジェクト指向クラス集合のクラス図上で, 121 の抽象化ルールを用意して，重要度の高いクラスのみを残し、抽象化する手法を提案している[9]．Egyed らによって公開されている抽象化ツールを，図 4 のサンプルシステムに対し適用したところ, 図 10 のグラフが得られた．このグラフに対して CCOF 値を求めると，以下を得る．

$$CCOF(\{E_1, \ldots, E_{16}\}) = \frac{17}{16^2 - 16} \quad 0.0708$$

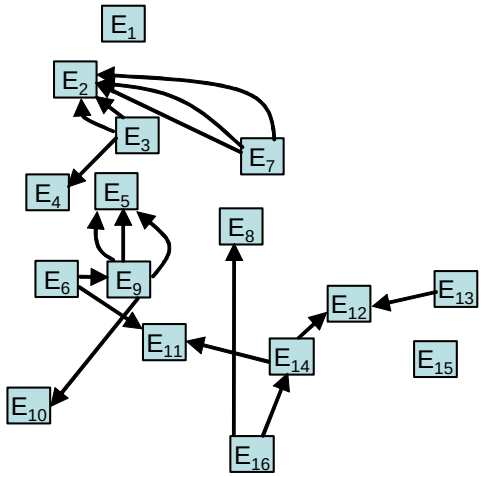

図10　Eygedのツールによる抽象化

図 10 を本稿で提案する抽象化に基づくグラフ図 6 と比較すると，EJB 実装クラスに対する関連が存在せず，保守性と複雑さが適切に関連していない．また，Eyged の抽象化手法は，リモートコンポーネントの実装に関わる特性（コンポーネントの実装クラスとインターフェースがクラスレベルでは関連していないこと）を考慮しない．

また，Egyed の抽象化手法以外にも，システム全体の理解と部分再利用を目的として，様々な抽象化手法・クラスタリング手法が提案されている．Doval らは，遺伝的アルゴリズムを用いた最適クラスタの決定という手法を提案している[11]．また，Girard らは，システムを構成するモジュール間の支配解析に基づくクラスタリング手法を提案している[12]．今後，これらの手法と CCOF の抽象化手法を比較し，併用する可能性について検討する予定である．

Bandi らは，オブジェクト指向クラス間の関係による複雑さは考慮せず、主にクラス単体の内部の複雑さを測定し、その合計をシステムの複雑度とする 3 つの複雑度測定法を提案している[13]．対して本稿が提案する抽象化手法は，クラス内部のクラスの複雑さは考慮せず、クラス間の関係から得られるグラフを抽象化することでコンポーネントベースシステム全体の複雑度測定法を適用している．

9. おわりに

実装アーキテクチャとして EJB を対象として，リモートコンポーネントを組み合わせて得られる CBS の複雑度を，リモートコンポーネントの実装に関わる特性を考慮した段階的抽象化によって測定する測定法 CCOF を提案した．リモートコンポーネントベースシステムの保守作業はコンポーネントを単位として行うことから，CCOF によって得られる複雑度はシステムの複雑さを適切に反映していると考えられる．測定実験の結果，CCOF がコンポーネントベースシステムの複雑度が機能拡張後の保守作業において有効であったため，保守において役立つ指標となる可能性がある．また，既存の測定法とは，独立していることを確認した．今後は，測定に用いるサンプル数を拡充し，実験結果に対する人手による更なる検証を行い，保守性との関連性を深めていく予定である．

また，EJB 以外のアーキテクチャについて，コンポーネントの特定手順を変更して実行することによって，CCOF を適用し測定可能とする予定である．

文　献

[1] 鈴木正人,丸山勝久,青木利晃,鷲崎弘宜,青山幹雄. コンポーネントウェア技術の確立に関する調査研究報告書,財団法人ソフトウェア工学研究財団, 産業技術研究所(2003)

[2] H.Washizaki, H.Yamamoto and Y.Fukazawa. A Metrics Suite for Measuring Reusability of Software Components, Proc. 9th IEEE Inter. Sympo. Software Metrics(2003)

[3] I. Singh, et al. Designing Enterprise Applications with the J2EE Platform,2nd Edition, AddisonWesley(2002)

[4] L.C.Briand, et al. A Unified Framework for Coupling Measurement in Object-Oriented Systems, IEEE Trans. Soft. Eng., Vol.25, No.1(1999)

[5] F.Abreu at al. Toward the Design Quality Evaluation of Object-Oriented Software Systems, Porc.5th Inter. Conf. Software Quality(1995)

[6] H.Washizaki and Y.Fukazawa. A Technique for Automatic Component Extraction from Object-Oriented Programs by Refactoring, J.Science of Computer Programming(2004, to be published)

[7] 松原望,統計の考え方,日本放送出版協会(2000)

[8] 阿萬裕久,山崎健司,山田宏之,野田松太郎. 主成分･相関分析によるメトリックスの定量的検証法, 電子情報通信学会論文誌, Vol.J35-D-I, No.10(2002)

[9] A.Egyed. Automated Abstraction of Class Diagrams, ACM Trans. Soft. Eng. and Methodology. Vol.11, No.4(2002)

[10] L. C. Briand, S. Morasca and V.R. Basili. Property-Based Software Engineering Measurement, IEEE Trans. Soft. Eng., Vol. 22, No. 1(1996)

[11] D. Doval, S. Mancoridis and B. S. Mitchell. Automatic Clustering of Software Systems using a Genetic Algorithm, Proc. International Conference on Software Tools and Engineering Practice (1999)

[12] J. F. Girard and R. Koschke. Finding components in a hierarchy of modules: A step towards architectural understanding, Proc. 13th International Conference on Software Maintenance (1997)

[13] R. K. Bandi and V. K. Vaishnavi. Predicting Maintenance Performance Using Object-Oriented Design Complexity Metrics, IEEE Trans. Soft. Eng., Vol. 29, No. 1 (2003)