バイナリコンポーネントベースソフトウェアの構造の機能規模を伴う視覚化

Visualization of Binary Component-based Software Structure with Component Functional Size

鷲崎 弘宜 † 高野 悟 †† 深澤 良彰 ††

Hironori WASHIZAKI Satoru TAKANO Yoshiaki FUKAZAWA

† 国立情報学研究所/総合研究大学院大学

National Institute of Informatics / The Graduate University for Advanced Studies

†† 早稲田大学理工学部コンピュータ・ネットワーク工学科

Dept. of Computer Science, School of Science and Engineering, Waseda University

washizaki@nii.ac.jp tsatoru_113@asagi.waseda.jp fukazawa@waseda.jp

開発コストの削減とソフトウェアの信頼性の向上を目的とした開発手法として，コンポーネントベース開発がある．コンポーネントベース開発ではしばしば，第三者によって開発されたソースコードを伴わないバイナリ形式のコンポーネントを再利用して，新たなソフトウェアを迅速に構築する．得られるソフトウェアを継続して保守するためには，ソフトウェア全体の構造/振る舞いの視覚化が重要である．しかしながら，従来のソフトウェア視覚化手法はプログラムソースコードの解析を必要とするため，バイナリコンポーネントを組み合わせて得られるソフトウェアへ適用することは難しい．本稿では，ソースコードを伴わない JavaBeans コンポーネントより構成されるプログラムについて，リフレクションとバイトコード解析の両技術を用いてコンポーネントおよび他の補助的クラス間の依存関係を取得し，各コンポーネントの機能規模の測定結果に基づいて，全体構造を適切に視覚化するシステムを提案する．

1 はじめに

オブジェクト指向開発における部品化の考え方を発展させて，より迅速に多様／大規模なソフトウェアを効率よく開発する手法として，コンポーネントベース開発がある [1, 2]．コンポーネントベース開発ではしばしば，第三者によって開発された，ソースコードを伴わずにバイナリコードの形式で配布されるコンポーネントを再利用して，新たなソフトウェアを迅速に構築する．そのようなコンポーネントを以降においてバイナリコンポーネントと呼ぶ．

ソフトウェアの保守作業について，ソフトウェアの理解に多くの時間が費やされることがよく知られている [3]．そこで，コンポーネントベース開発に従って得られるソフトウェアを効率よく継続的に保守するためには，コンポーネントの集合として構成される対象ソフトウェア（コンポーネントベースソフトウェア）の様々な側面に関する大量の情報を主に図表現によって視覚化し，保守作業従事者による直感的な理解を促す必要がある．

ソフトウェアの主要な側面として静的構造と動的振舞いの 2 種類があるが，本稿では前者を対象とし，コンポーネントベースソフトウェアの静的構造の視覚化について議論する．従来のソフトウェア構造の視覚化手法/システム（例えば SEIV[3] や Seesoft[4], ObjectOrrery[5] など）は，主にプログラムソースコードの解析を必要とするため，バイナリコンポーネントを組み合わせて得られるソフトウェアの視覚化に利用することは困難である．

本稿では，ソースコードを伴わない JavaBeans[6] バイナリコンポーネントより構成されるプログラムについて，リフレクションとバイトコード解析の両技術を用いて，コンポーネント（および他の補助的クラス）間の依存関係を取得して，視覚的に図表現によって表示するシステムを提案する．JavaBeans は，オブジェクト指向プログラミング言語 Java の上で，コンポーネントを開発し利用するためのコンポーネントアーキテクチャである．さらに，提案するシステムは，我々が独自に定義するコンポーネント機能規模測定法の適用により得られる機能規模測定値を活用することで，プログラムを構成する個々のコン

ポーネントの機能規模を分かりやすく表現する．

2 コンポーネントベース開発と JavaBeans

コンポーネントとは，ある機能を提供する，独立して交換/再利用可能なソフトウェア構成単位である [1]．コンポーネントは一般に，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 [7]．開発基盤となるソフトウェアアーキテクチャ（コンポーネントアーキテクチャ）を決定し，決定したアーキテクチャの規格に従って動作可能なコンポーネントを再利用，もしくは新規に開発し，得られるコンポーネントを組み合わせて新たなソフトウェアを開発する手法をコンポーネントベース開発と呼ぶ．

コンポーネントアーキテクチャは，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 [8]．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，同一環境上に配置するものを指す．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，異なる環境上に配置するものを指す．

これまでに，視覚表現に基づくコンポーネント組立て型開発環境の充実，および，Web アプリケーション実現環境の普及（JSP，ASP など）に伴い，GUI 部品や汎用的なロジック部品を中心としてクライアントコンポーネントが先に普及してきた．そこで本稿では，視覚化の対象として，JavaBeans コンポーネント [6] の組み合わせによって構築された Java プログラムを扱う．

JavaBeans は，Java 言語上でローカルコンポーネントを開発し利用するためのコンポーネントアーキテクチャである．JavaBeans に従ったコンポーネントを Bean と呼ぶ．Bean は，以下の条件 1 ～ 2 を満たす Java 言語上の 1 つのクラスとして定義される．従って Bean は，通常の Java クラスとしての構成要素であるコンストラクタ，フィールド，メソッドを持つ．

- 条件 1: 外部から呼び出し可能な引数を伴わないコンストラクタを持ち，クラス名のみの指定によってインスタンスを生成可能である．
- 条件 2: `java.io.Serializable` インタフェースを実装し，直列化可能である．

例として，典型的な Bean の静的構造を UML クラス図として図 1 に示す．この例においてクラス `Chart` は，JavaBeans の定義に従った Bean である．

JavaBeans は上述の定義に加えて，開発環境や他の Bean から機能を扱いやすくするために，対象クラスおよび関連クラスが以下の仕組みに従うことを推奨する．

- プロパティ: Bean の外部から値を取得あるいは設定可能な名前付きの特性をプロパティと呼ぶ．プロパティは，Bean として扱う対象クラスについて，外部から特性の値を設定可能とする設定メソッドと，外部から特性の値を取得可能とする取得メソッドを実装することによって定義される．両メソッドをプロパティアクセスメソッドと呼ぶ．プロパティアクセスメソッドの実装は，主に命名規則とメソッドの型付けに従って行われる．具体的には，対象クラスが，型 `A` を戻り値とするメソッド `public A getXyz()`（型 `A` を引数とするメソッド `public void setXyz(A a)`）を持つときに，値を設定可能な（値を取得可能な）プロパティ `xyz` が存在すると認識できる．Bean が持つプロパティの多くは，Bean としてのクラスに実装されたフィールドに一対一に対応する傾向にある [15]．図 1 の例では，クラス `Chart` は，フィールド `title` の値を設定/取得するメソッド `setTitle()`/`getTitle()` を持つ．従って Bean としての `Chart` は，値を設定かつ取得可能なプロパティ `title` を持つ．
- メソッド: Bean が外部に対して提供する機能をメソッドと呼ぶ．メソッドは，Bean として扱う対象クラスについて，外部から呼び出し可能な（public な）メソッドを実装することによって定義される．図 1 の例では，`Chart` はメソッド `plot()` を持つ Bean である．
- イベント: Bean 内で何らかの事象が起きたことを外部に知らせる仕組みがイベントである．イベントを構成する要素は，イベントソース，イベントリスナ，イベントオブジェクトである．イベントソースは，事象の発生した対象 Bean を指す．イベントリスナの登録/登録解除メソッド，イベントオブジェクトクラス，および，イベントリスナの実装は，主に命名規則と型付けに従って行われる．図 1 の例では，`Chart` はイベント `Updated` を持つ Bean である．

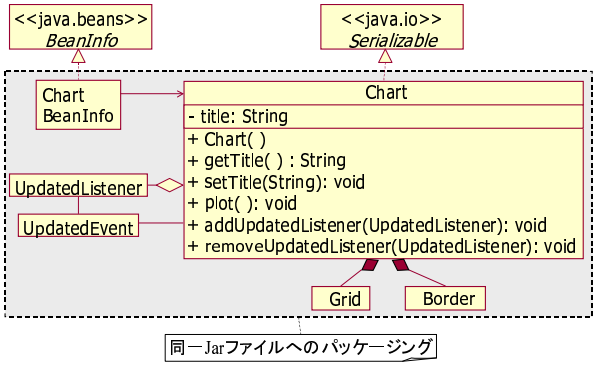

図 1: Bean および関連クラスの例（クラス図）

上述の定義と仕組みは，Bean を単独で配布したときに，Bean が依存する他のクラス/インタフェース（例えばイベントリスナ）が配布された環境に存在することを保証しない．そこで，アーカイブファイル形式 JAR を利用して，Bean が依存する全ての Java クラス/インタフェースを同一のアーカイブファイルに格納して配布/再利用することが推奨される．図 1 の例では，Bean としての `Chart` が依存する他のクラス `Grid`，`Border` およびイベント関連のクラス/インタフェースを全てまとめて一つの Jar ファイルに格納して配布する必要がある．

バイナリ形式（Java バイトコード）で提供される Bean について，リフレクション機構（およびイントロスペクション機構）を用いて，ソースコードを解析せずに以下の情報 1~3 を外部から取得できる．

1. Bean としての基本情報: プロパティ，イベント，メソッドに関する情報を，イントロスペクション機構 [6] を利用して取得できる．それらの情報は，命名規則，もしくは，Bean 開発者が独自のメタ情報として用意する `BeanInfo` オブジェクトを解析することで得られる．

2. クラスとしての構成要素の情報: Bean がクラスとして持つコンストラクタ，フィールド，メソッドに関する情報をリフレクション機構を利用して取得できる．

3. アーカイブファイル構成情報: Bean が含まれるアーカイブファイルの構成情報（アイコンなどの各種リソースファイルの有無，Bean が依存する他のクラス/インタフェースの構成要素情報）も，ソースコードを解析せずに外部から取得できる．

一方，以下に挙げる情報 4 は，Java のリフレクション機構，および，JavaBeans のイントロスペクション機構によって取得できない．以下の情報 4 をコンパイル後の Java バイトコードから取得するためには，直接的なバイトコード解析 [11] を行う必要がある．

4. クラス間の依存関係: Bean としての Java クラスが，他の Bean や補助的なクラスを利用/インスタンス生成する，あるいは，利用/インスタンス生成される関係を，バイトコード解析によって取得できる．

Bean の機能規模を測定するにあたって，これらの外部から取得可能な公開情報 1~4 が，その重要な判断材料となる．

3 ソフトウェアの視覚化

本章では，我々が試作したコンポーネントベースソフトウェア視覚化システムの仕組み/機能を議論するために，視覚化システム一般に共通する仕組みと，プログラム視覚化システムについてそれぞれ詳説する．

3.1 視覚化システム参照モデル

Card らは，視覚情報に限定されずに，対象世界に内在する構造/傾向などの情報を抽象化し視覚化/表示するシステム（視覚化システム）一般に共通する事柄/仕組みを，参照モデルとして提案している [9]. 同参照モデルは，視覚化システムを，扱うデータ表現の形態が以下に挙げる 4 つの状態を遷移するモデルとして定義する．

- ローデータ（生データ）: 視覚化して理解/認識する情報の基礎/元となるデータ
- データテーブル: ローデータから抽象化/変換を通じて抽出された，視覚化のために必要な情報
- 視覚構造: データテーブルから，特定の空間上に配置/マッピングされた情報
- 表示情報: 視覚構造から，実際に利用者が用いる特定の表示機器（例えば 2 次元ディスプレイ）上で提示するために変換された情報

これらの状態の間，および，システムと表示情報を閲覧する利用者の間に発生する相互作用を以下に示す．

- データ変換: 視覚化する情報に対する変更
- 視覚的マッピング: 視覚化の方法に対する変更
- ビュー変換: 視覚化された情報に対する変更

次章において，我々が試作したシステムの仕組みを，上述の参照モデルに従って述べる．

3.2 プログラムの視覚化

ソフトウェア開発を支援するために，文字列形式の最終ソフトウェア（プログラム）の構造や振る舞いなどを視覚的に表す行為を，プログラムの視覚化と呼ぶ．プログラムの視覚化は，視覚化対象（プログラム，または，プログラムが扱うデータ）と，描画方法（静的，または，動的）の組み合わせより 4 種に分類できる [10]．それらのうちで，本稿ではコンポーネントベースソフトウェアを視覚化の対象とした静的プログラム視覚化システムを扱う．

これまでに，オブジェクト指向プログラムを視覚化の対象として，Seesoft[4] や ObjectOrrery[5] に代表される種々の静的プログラム視覚化システムが存在する．例えば ObjectOrrery は，オブジェクト指向プログラミング言語 Smalltalk によって記述されたクラス群における参照関係の集合から部分集合を抽出して表示する機能や，クラス構造について特定の変更の影響が及ぶ範囲を表示する機能を持つ．

しかしながら，従来のプログラム視覚化システムは，対象とするプログラムソースコードの解析を必要とするため，バイナリ形式で提供されるコンポーネントより構成されるバイナリコンポーネントベースのプログラムに適用困難である．また，プログラムを構成する個々のコンポーネントの特性（例えば機能規模）を，扱うコンポーネントアーキテクチャに応じて表示する機能を有さない．

4 バイナリコンポーネントベースソフトウェアの視覚化

ソースコードを伴わず，バイナリコードのみ提供されるコンポーネントおよび補助的クラスを組み合わせて得られる最終ソフトウェア（プログラム）を，バイナリコンポーネントベースソフトウェアと呼ぶ．我々は，開発者による拡張や修正といった保守作業の効率の良い実施のために必要なソフトウェア理解の支援を目的として，バイトコード形式で提供される JavaBeans に基づいたコンポーネント（Bean）や，補助的な Java クラスを組み合わせて実装された Java プログラムの静的構造を，視覚的に表示するシステム（以降，本システム）を提案する．

本システムは，リフレクション/イントロスペクション機構およびバイトコード解析の活用により，Bean およびクラス間の依存関係と，各 Bean の機能規模を視覚化する．

4.1 システム構成

本システムの構成を図 2 に示す．上述の視覚化参照モデルに基づき，本システムにおけるデータ表現に関する状態遷移と相互作用を以下に述べる．

- ローデータ: Java バイトコードの集合
- データテーブル: バイトコードの集合より得られるクラス/コンポーネント間の依存関係，および，各コンポーネントの機能規模
- 視覚化構造: 3 次元空間に配置された依存関係/機能規模
- 表示情報: 計算機ディスプレイにおいて表示できるように 2 次元平面上に（視覚化構造からの変換を通じて）配置された依存関係/機能規模
- データ変換: コンポーネント解析部と依存解析部の両解析部により，ローデータからデータテーブルを構築する．前者は，対象とする Java クラスが前述の JavaBeans の仕様を満たしていることを判定し，Bean であると判定した場合は，リフレクション機構を用いて機能規模を測定する．後者は，バイトコード解析器として Javassist[11] を利用して，Java バイトコードにおけるコンスタントプール内の情報を解析し，クラス間の依存関係を取得する．両解析部の仕組みについては次節以降において詳説する．
- 視覚的マッピング: ローデータを構成するクラス，コンポーネント，および，それらの間の依存関係を視覚化オブジェクト（ローデータにおける視覚化の単位）として，3 次元グラフィックス/マルチメディアフレームワークである”じゅん for Java”[12] を用いて 3 次元空間上の座標に配置する．クラス/コンポーネントに対応する視覚化オブジェクトは，それぞれの特性情報（名称や機能規模）を持つ．

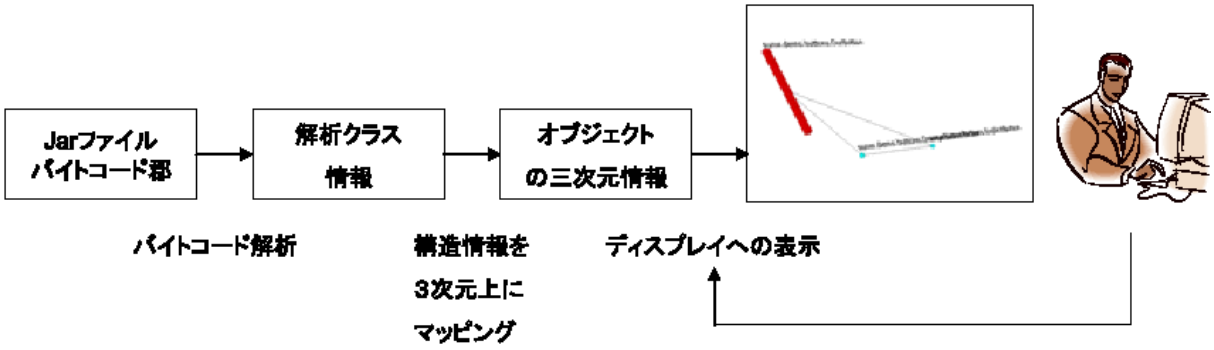

図 2: バイナリコンポーネントベースソフトウェア視覚化システムの構成

- ビュー変換: 視覚的マッピングによって得られる 3 次元座標データを，”じゅん for Java”を用いて 2 次元画像として出力する．

4.2 依存解析部

依存解析部はバイトコード解析によって，以下に挙げるクラス間の依存関係（継承関係，参照関係，および，インスタンス生成関係）を取得する．

- 継承関係: あるクラスが Java 言語における `extends` 句を用いて他のクラスを拡張して定義されている関係を，継承関係と呼ぶ．
- 参照関係: あるクラスが他のクラスの型/オブジェクトを利用している（例えば，オブジェクトを値として用いたり，オブジェクトのメソッド/フィールドを利用している）関係を，参照関係と呼ぶ．
- インスタンス生成関係: あるクラス（または同クラスのオブジェクト）が，内部において他のクラスのオブジェクトを生成（`new`）している関係を，インスタンス生成関係と呼ぶ．本システムはインスタンス生成関係を，クラス間の”強い”関係として位置づけて，上述の参照関係と区別して捉える．

4.3 コンポーネント解析部

コンポーネント解析部は，2 章で述べた定義に従って通常の Java クラスと Bean を判別し，対象とするクラスが Bean であると判定した場合は，リフレクション機構を用いて機能規模を測定する．コンポーネントの機能規模とは，対象コンポーネントが提供する機能の大きさである．コンポーネントベースソフトウェアを構成する各コンポーネントの機能の大きさがどの程度のものであるのかを視覚的に表すことにより，同ソフトウェア全体における機能分割/割り当ての妥当さや，全体の機能の大きさの程度を直感的に理解することを支援できる．

コンポーネントの機能規模に関わる公開された特性として，対象コンポーネントが持つメソッド数，プロパティ数，および，イベント数を考えることができる．

- メソッド数: コンポーネントが保有するメソッドの数は，同コンポーネントが提供する機能のうちで，主に同コンポーネントの内部に作用する機能の規模を反映する．
- イベント数: コンポーネントが保有するイベントの数は，同コンポーネントが提供する機能のうちで，主に同コンポーネントの外部（つまり他のコンポーネントやクラス）に作用する機能の規模を反映する．
- プロパティ数: コンポーネントが提供するメソッドは一般に，その実行時に特定のプロパティの値を利用し，実行結果を特定のプロパティに値として設定する [15]．また，コンポーネントが提供するイベントは一般に，特定のプロパティの値の違いに応じて実際に発火し外部に通知される．従って，コンポーネントが保有するプロパティの数は，同コンポーネントの内部に作用する機能の入力/出力に関する規模，および，外部に作用する機能の起動条件に関する規模を反映する．

コンポーネント解析部は，イントロスペクション機構を用いてこれらの 3 種の値を測定する．ただし，

これらの値は上述のように，それぞれ異なる観点からコンポーネントの機能規模を表している．そこで，これらの値を結合することによって，コンポーネント c の総合的な機能規模を表す測定法 $FOC(c)$（Functional size Of Component）を以下に定義する．ここで，$NOM(c)$, $NOE(c)$, $NOP(c)$ はそれぞれ c のメソッド数，イベント数，および，プロパティ数を表すものとする．

$$FOC(c) := NOM(c) \cdot NOE(c) \cdot NOP(c)$$

ここで，コンポーネントの機能規模 FOC の測定値は，欠陥数や複雑さといった（意味について）単純な測定法とは異なり，値が大きいほど対象コンポーネントが優れる（あるいは優れない）と解釈できるものではない．つまり，バイナリ形式で提供され内部が隠蔽されたコンポーネントについて機能規模が大きいほど，対象コンポーネントの再利用効果は高くなるが，同コンポーネントの機能を理解するための労力はより大きなものとなり保守上の問題を抱える可能性は増大する．

そこで，コンポーネントベースソフトウェアを構成するバイナリコンポーネントの機能規模の適切な基準値を求める．我々は，提案した FOC の基準値を求めるにあたり，JARS.COM[13] において公開されている人手によるコンポーネントの評価情報を使用した．JARS.COM における評価情報は，これまでに幾つかの測定法の基準値の設定に使用されている実績を持つ [14, 15, 16, 17]．JARS.COM では，多人数で長時間をかけて，Programming や Science といったカテゴリごとに大量の Bean の表現性/機能性/新規性を評価し，8 段階評価で評価付けを行っている．この 8 段階評価を区間 $[0, 1]$（1 が最も良い）に収まるように正規化し，その値を JARS 評価と定義する．我々は，2005 年 3 月の時点で JARS.COM において JARS 評価が与えられている全 118 個の Bean を評価サンプルとして用いた．JARS.COM における Bean の公開は，不特定多数によって再利用されることを意味するため，JARS 評価は Bean が再利用されることを考慮しており，コンポーネントの機能規模を含む品質全体の高さを反映していると考えられる．

基準値の算出方法として，最初に，JARS 評価が 1 の群（A 群，コンポーネント数: 95）と JARS 評価が 1 未満の群（B 群，23）に分け，A 群/B 群の FOC の平均値を算出した．両群のカテゴリごとの JARS 評価の平均値，および，カテゴリによらない平均値を表 1 に示す．

我々は，表 1 において得られた A 群の平均値を，コンポーネントの機能規模の適切な基準値の上限として採用し，FOC の測定値が同上限以内である場合に測定対象コンポーネントが機能規模について適切であると判定する．

表 1: 両群におけるカテゴリごとの FOC 平均値

カテゴリ	Bean の数	FOC の平均値	
		A 群	B 群
Programming	113	188092	2714166
Utilities	2	253760	0
Game	2	–	182003
Science	1	–	645932
全平均（合計）	（118）	188783	2286048

4.4 視覚化オブジェクト

本システムは，視覚的マッピングを行う際に，以下の要素をそれぞれ視覚化オブジェクトとして表示する．

- コンポーネント: コンポーネントの視覚化オブジェクトを，FOC の測定値を反映した直方体として表示する．具体的には，プロパティ数/メソッド数/イベント数を，それぞれ直方体の縦/横/高さに対応付けて，直方体の体積としてコンポーネントの機能規模 FOC を示す視覚化オブジェクトを作成する．

  さらに，FOC の基準値を用いて機能規模の適切さを判定し，機能規模について適切/不適切なコンポーネントをそれぞれ異なる色の異なる視覚化オブジェクトとして表現する．具体的には，FOC の測定値が同基準値以内のコンポーネントは赤色の直方体として，同基準値を超えるコンポーネントは黒色の直方体としてそれぞれ表示する．

- クラス: コンポーネント（Bean）ではない Java クラスについては，青色の球体として視覚化オブジェクトを表示する．

- クラス/コンポーネント間依存関係: クラス/コンポーネント間の依存関係は，クラス/コンポーネントにそれぞれ対応する視覚化オブジェクト間の直線によって表示する．

本システムの実行例として，JavaBeans 対応開発環境参照実装 BeanBox[18] における動作が確認され

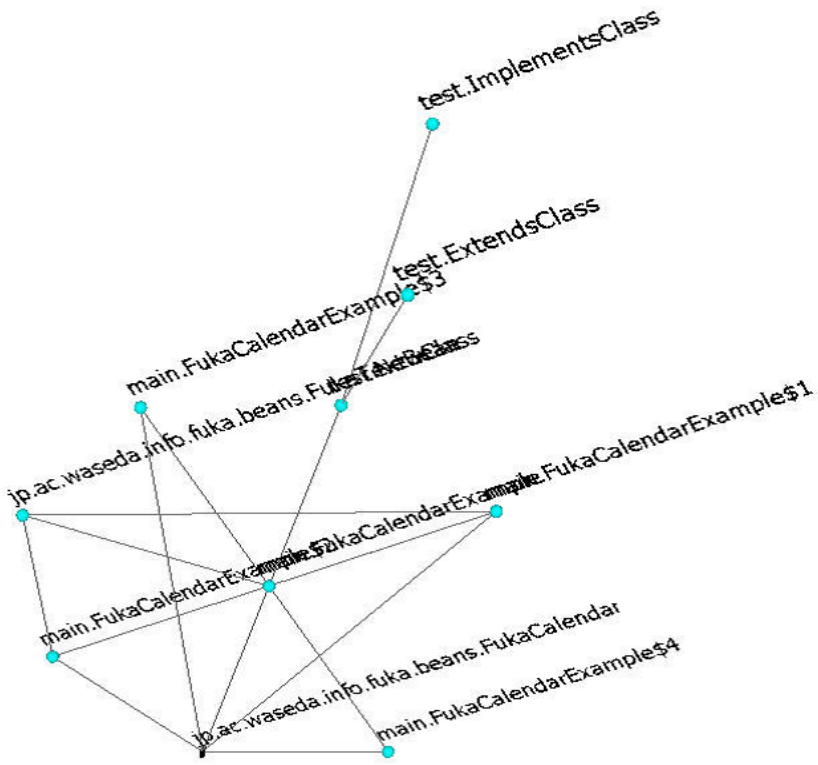

図 3: 本システムによるバイナリコンポーネントベースソフトウェアの視覚化例

ている JavaBean バイナリコンポーネントライブラリ”FukaBeans”[19] を用いて実装されたバイナリコンポーネントベースソフトウェアを入力して，視覚化した様子を図 3 に示す．本システムによって同ソフトウェアを解析し視覚化した結果，依存解析とコンポーネント機能規模測定に基づいて，コンポーネントやクラスの特性，および，それらの関係が適切に視覚化されることを確認した．例えば図 3 において，コンポーネント `FukaCalendar` は，他のクラスとは区別され，機能規模を反映した直方体として表示されている．

5 システムの比較評価

SeeSoft や ObjectOrrey に代表される従来の静的プログラム視覚化システムと本システムを，視覚化可能な情報について比較した結果を表 2 に示す．表 2 において，従来の視覚化システムは対象プログラムを構成するクラスとコンポーネントを区別せず共通に表示するため，利用者は視覚的にクラス/コンポーネントを判別することができず，コンポーネントベースソフトウェアの内部構造の直感的な理解の支援には適さない．また，従来の視覚化システムはソースコードの解析を前提とし，一部についてソースコードが得られないバイナリコンポーネントベースソフトウェアを扱うことができない．

対して本システムは，クラス/コンポーネントを区別した表示を行い，コンポーネントについてはその機能規模を視覚的に表すため，利用者がバイナリコンポーネントベースソフトウェアの内部構造を直感的に理解することを適切に支援する．

視覚化情報	本システム	従来システム
クラス	Y	Y
コンポーネント	Y	n
機能規模	Y	n
継承関係	Y	Y
参照関係	Y	Y
生成関係	Y	Y

表 2: 視覚化システムの機能比較（Y: 視覚化対象として扱う，n: 扱わない）

6 おわりに

本稿では，JavaBeans に従い，かつ，バイナリ形式で提供されるコンポーネントを構成要素として含むバイナリコンポーネントベースソフトウェアの構造を，リフレクション/イントロスペクション機構とバイトコード解析によって解析し視覚的に表示するシステムを提案した．提案システムは，クラスとコンポーネントの区別，クラス/コンポーネント間の依存関係，および，コンポーネントの機能規模を明示的に表すことにより，従来の視覚化システムと比較して，バイナリコンポーネントベースソフトウェアの直感的な理解をより適切に支援することができる．機能規模については，JavaBeans コンポーネントの機能規模に関係する複数の特性を考慮した測定法 FOC を独自に考案し，大量の既存コンポーネントより同測定法の適切な基準値を算出し，基準値と照らし合わせて表示する仕組みを採用した．機能規模を伴った視覚化によって，ソフトウェア全体における機能分割/割り当ての妥当さや，全体の機能の大きさの程度を直感的に理解することを支援できる．

今後の課題として，視覚化オブジェクトの配置方法の最適化，および，コンポーネントの機能規模以外の種々の特性を同時に表示する仕組みの確立が挙げられる．また，提案システムによる大規模なバイナリコンポーネントベースソフトウェアの視覚化とソフトウェア理解作業実験を通じて，提案システムの有効性を実証的に明らかにする予定である．

参考文献

[1] C. Szyperski: Component Software: Beyond Object-Oriented Programming, Addison-Wesley, 1999.

[2] C. Atkinson: Component-based Product Line Engineering with UML, Addison-Wesley, 2001.

[3] 鈴木宏紀, 山本晋一郎, 阿草清滋: プログラムの構造情報の視覚表現を用いた動作情報の視覚化, 日本ソフトウェア科学会第 6 回インタラクティブシステムとソフトウェアに関するワークショップ論文集（WISS'98）, pp.99–104, 1998.

[4] S. Eick, J. Steffen, and E. Summer, Seesoft: A Tool For Visualizing Line Oriented Software Statistics, IEEE Transactions on Software Engineering, Vol.18, No.11, pp.957-968, 1992.

[5] N. Nishikawa: ObjectOrrery: 3d Visualization of Class Library Structure, 3rd Workshop on Software Visualization, 1999.

[6] G. Hamilton: JavaBeans 1.01 Specification, Sun Microsystems, 1997.

[7] H. Washizaki and Y. Fukazawa: A Technique for Automatic Component Extraction from Object-Oriented Programs by Refactoring, Science of Computer Programming, Vol.56, No.1–2, pp.99–116, 2005.

[8] 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, ソフトウェア工学研究財団, 2003.

[9] S.K. Card, J.D. Mackinlay and B. Shneiderman: Information Visualization: Readings in Information Visualization Using Vision to Think, Morgan Kaufmann Publisher, Inc., 1999.

[10] B.A. Myers: Visual Programing, Programing by Example and Program Visualization, Proceeding of the ACM Conference on Human Factors in Computing System (CHI'86), pp.59–66, 1986.

[11] S. Chiba and M. Nishizawa: An Easy-to-Use Toolkit for Efficient Java Bytecode Translators, Proceedings of the 2nd International Conference on Generative Programming and Component Engineering (GPCE '03), Springer LNCS Vol.2830, pp.364–376, 2003.

[12] じゅん for Java, `http://www.sra.co.jp/people/nisinaka/Jun4Java/`

[13] JARS.COM, http://www.jars.com/

[14] 山本浩数, 鷲崎弘宜, 深澤良彰: 再利用特性に基づくコンポーネントメトリクスの提案と検証, コンピュータソフトウェア, Vol.19, No.5, pp.13–25, 2002.

[15] H. Washizaki, H. Yamamoto and Y. Fukazawa: A Metrics Suite for Measuring Reusability of Software Components, Proceedings of the 9th IEEE International Symposium on Software Metrics, pp.211–223, 2003.

[16] 平山雅之, 佐藤誠: ソフトウェアコンポーネントの利用性評価, 情報処理学会論文誌, Vol.45, No.6, 2004.

[17] 鷲崎弘宜, 平口裕紀, 深澤良彰: JavaBeans コンポーネントの品質メトリクスの提案, 情報処理学会ソフトウェアジャパン 2004 論文集, 2004.

[18] Sun Microsystems, BeanBox, `http://java.sun.com/products/javabeans/`

[19] 早稲田大学深澤研究室: FukaBeans, `http://www.fuka.info.waseda.ac.jp/Project/CBSE/`