ソフトウェア工学 137－4
（2002．5．10）

システム理解のための分散システムアーキテクチャの抽出

方 学芬 †,†† 玉井哲雄 ††

計算機ネットワークの発展に伴い，複数の計算機を結合した分散アプリケーションシステムの構築が盛んに行われている．これらのアプリケーションシステムは企業のビジネスと関わりが深く，システムを安全に，かつ一貫性を保ちながら保守することが要求される．質が高い保守を行うためには，ソフトウェアシステムの理解は必須である．アプリケーションの様々な特性の多くはそのアーキテクチャによって定まるために，複雑なアプリケーションシステムを理解し保守するにはアーキテクチャ情報が欠かせない重要な情報である．

しかし，これらの情報は十分にドキュメント化されていない場合が多い．本稿では，最初に分散アプリケーションシステムを対象に，メンテナンスに必要とする具体的なアーキテクチャを概要アーキテクチャ，機能写像アーキテクチャ及び配置アーキテクチャに分けて記述する．次に概念的アーキテクチャに基づき，具体的なアーキテクチャを抽出するプロセスを示す．最後に具体的な分散アプリケーションシステム E-Process に対して，3 種類のアーキテクチャを抽出することにより，その方法が有効であることを確認する．

Keywords コンポーネントベースの分散システム，アーキテクチャ抽出，メンテナンス

Abstracing Architecture of Distributed System for System understanding

XUEFEN FANG †,†† and TETSUO TAMAI ††

Many software systems do not have a documented concrete architecture. These are often large and complex systems that are difficult to understand and maintain. One approach to understanding of a system is to extract architectural documentation from implemented system. We propose three types of concrete architecture for component-based distributed application system. They are functional architecture, functionality mapping architecture and configuration architecture. To evaluate the effectiveness of this approach, we extracted architectural documentation from three tiers distributed system E-process. Our study shows that three types of architecture are useful for understanding and maintaining the E-Process system structure.

1. はじめに

ソフトウェアシステムは，現実の社会に組み込まれて動くものが増加し，個人生活のツールから企業のビジネスツールまで幅広く使われている．そのようなソフトウェアシステムは，現実の社会システムの変化に応じて常に進化なければいけないことが，早くも 20 年前に Lehman 達によって指摘されている[7]．ソフトウェア技術の進歩に伴い，システムが置かれている計算環境の変化によって，ソフトウェアシステムの進化が要求される．このような進化を促す作業は一般的にメンテナンスといわれている．メンテナンス

† SRA 先端技術研究所
SRA Key Technology Laboratory, Inc.
†† 東京大学大学院総合文化研究科広域システム専攻
Dept. of Graphics and Computer Science Graduate School of Arts and Sciences The University of Tokyo

の中でシステムの調査分析または理解に要する作業は45%-55%の割合を占める．質が高いメンテナンスを行うためには，ソフトウェアシステムの理解は重要な技術であると認識されている．

計算機ネットワークの発展に伴い，複数の計算機を結合した分散アプリケーションシステムの構築が盛んに行われている．これらのアプリケーションシステムは，企業のビジネスと関わりが深く，システムを安全に，かつ一貫性を保ちながら保守することが要求される．そのためには，ソフトウェアアーキテクチャを同定してシステムの様々な特性を把握することが必須である．

しかし，現実的にアプリケーションシステムの設計段階で実装されたアーキテクチャは，十分にドキュメント化されていないことが多い．抽象度の高い概念的なアーキテクチャの記述があったとしても，システム理解やメンテナンスのためには，不十分であることが多い．またソフトウェアシステムの変更に伴って，既存のドキュメントとソースコードとの不一致が生じやすく，結果的にアプリケーションシステムの理解がソースコードベースとなるのが現実である．

大規模なシステムに対して，膨大なソースコードを解読することはとても手間がかかる．それを支援するために，既存システムのソースを解析し，データの参照関係，モジュール間の呼び出し関係などの情報を作成，表示するリバースエンジニアリング技術がある．従来の分散していないシステムを理解するために，システム共有変数の参照情報，モジュール間の呼び出し関係をソースコードから逆に構成して，可視的に表現することによって，既存システムの理解を深める研究が盛んに行われた[9),11),15)]．また変数の参照関係というレベルより抽象度を高めたレベルで理解するために，ソフトウェアアーキテクチャの重要性が認識され，アーキテクチャ抽出についての研究も行われている[6),10)]．これらの研究の重点は，システムのモジュール呼び出し関係によるアーキテクチャ抽出で，多くのファイルやモジュールから，ネーミングルールを利用して高レベルのサブシステムを形成することにより，サブシステムレベルのアーキテクチャが構成される．代表的なアーキテクチャの抽出例として，OpenSource の Mozilla ブラウザ，Linux などのシステムのアーキテクチャを抽出する実例が報告されている[4),14)]．

しかし，これらの手法は，主に逐次的なシステムを対象としたものであり，増え続けるコンポーネントベースの分散システムを対象とするアーキテクチャ抽出に対応していない．本研究は分散アプリケーションシステムを対象に，メンテナンスに必要となる具体的なアーキテクチャを機能アーキテクチャ，機能写像アーキテクチャ及び配置アーキテクチャに分類し定義する．また概念的アーキテクチャに基づき，これらのアーキテクチャの抽出方法を検証する．事例として，EJB(Enterprise JavaBeans)コンポーネントモデルを基にした分散3階層アプリケーションシステム E-Process☆に対して，3種類のアーキテクチャを抽出することにより，この方法が有効であることを確認した．

本文の構成は以下のとおり，2章ではコンポーネントベースの分散アプリケーションシステムの特徴を述べる．3章では概念的なアーキテクチャと具体的なアーキテクチャを定義する．4章では具体的なアーキテクチャ抽出プロセスを述べ，5章では E-Process の具体的なアーキテクチャの抽出事例を紹介し，6章で結論づける．

2. コンポーネントモデルベースの分散アプリケーションシステムの特徴

分散アプリケーションシステムは，様々な分散環境で構築される．本稿での分散アプリケーションは，主に分散コンポーネントモデルに基づいて構築されるアプリケーションを対象とする．分散コンポーネントモデルの代表例として COM/DCOM/COM+，OMG CORBA，.NET，EJB(Enterprise JavaBeans)などがある．このようなコンポーネント基盤の上で，既存コンポーネントを組み合わせることによって，アプリケーションシステムが柔軟に設計できると期待されている．アプリケーションシステムの分散配置によって大規模システムの実現が可能になり，またソフトウェアのコンポーネントの再利用の促進や将来の拡張性の確保が期待できる．一方コンポーネントの情報隠蔽，動的な結合などの特性により，従来のアプリケーションシステムを理解する方法ではコンポーネントモデルベースの分散アプリケーションシステムには適用できず，

☆ SRA&ASTI 上海が共同開発したソフトウェア開発プロセスサポートするシステム

メンテナンスに対応できない場合がある．例えば，分散協調処理が隠蔽されているため，基盤コンポーネントシステムの理解なしにアプリケーションシステムを理解することはできない．また基盤コンポーネントシステム上に実現する部分とアプリケーション固有部分として実現するものとの切り分けには，大きな自由度があるために，メンテナンス時の問題の特定が難しい．

ただし，分散アプリケーションシステムは次のような共通な特徴がある．

- コンポーネントモデルに依存する機能分割と配置に一定の規律性がある
- コンポーネントモデル間に標準通信プロトコルが採用される

これらの特徴により，概念的なアーキテクチャに基づいて，分散アプリケーションシステムの具体的なアーキテクチャを抽出できる．

3. 概念的なアーキテクチャと具体的なアーキテクチャ

ソフトウェアアーキテクチャは，ソフトウェアの構成要素と要素間の関係を表すものである．

概念的なアーキテクチャは，主に開発視点から捉えた抽象的なシステムの構成関係であり，開発視点から重要なサブシステムの関係を示している．

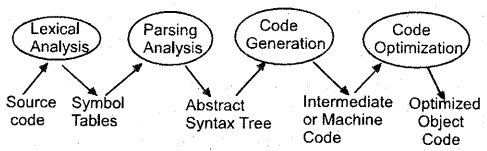

図 1　コンパイルアーキテクチャ

例えば，コンパイラの概念的なアーキテクチャは図 1 のように構成される．ここには基本的なサブシステムの要素しか含まれていないので，分かりやすい．このような概念的なアーキテクチャは具体的なシステムを理解するフレームワークとして使える．

具体的なアーキテクチャは，実装したシステムの構成要素と要素間の関係を表すものである．具体的なアーキテクチャは，またいくつかの角度から見ることができる．本稿では分散アプリケーションシステムの具体的なアーキテクチャを，機能アーキテクチャ，機能写像アーキテクチャ及び配置アーキテクチャに分けて記述する．この 3 つのアーキテクチャによって，分散アプリケーションシステムに対して次の情報を明らかにする．

- システムの全機能の構成
- 機能単位の固まり
- 機能からプログラミング領域への写像
- コンポーネントサーバへの配置

3.1 機能アーキテクチャ

機能アーキテクチャは，保守者の視点から見たアプリケーションシステムの機能モジュールの構造である．機能アーキテクチャには，ユーザの役割，ユーザインタフェースモジュール，ロジックモジュールとリソース (Database) 間の基本関係を表す．ユーザの役割はユーザがどのような立場でアプリケーションシステムを使うことを指す．機能アーキテクチャによって，どのような立場で，システムのどの機能モジュールが利用され，どのリソースにアクセスされるかを明らかにするシナリオが得られる．

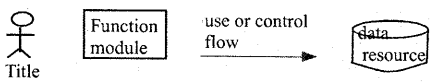

図 2　機能アーキテクチャの記号

機能アーキテクチャの記述について，図 2 に示す簡単な記号を用いて記述する．ユーザの役割を取り入れた機能アーキテクチャによって，メンテナンスの問題範囲を特定しやすくなる．

3.2 機能写像アーキテクチャ

機能写像アーキテクチャは，機能アーキテクチャに示した機能モジュールを概念的なアーキテクチャに対応するコンポーネントへの写像構造を表す．

機能モジュールの一つに対して，一つの機能写像アーキテクチャを抽出する．機能写像アーキテクチャには，指定した機能を実現する各階層でのコンポーネントとコンポーネント間の接続関係が表示される．コンポーネントを実行するコンテナの記述子の内容とプログラムソースにより，コンポーネント間の接続関係とコンポー

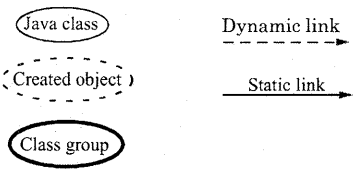

図 3　機能写像アーキテクチャの記号

ネントの属性などが確認できる．

機能写像アーキテクチャは図 3 に示す簡単な記号を用いて記述する．

3.3　配置アーキテクチャ

配置アーキテクチャは，実際のサーバに配置したコンポーネントと，それらのコンポーネントの構成関係を表す．具体的には，コンテナのディレクトリ構造とコンポーネントの構成関係を表す．配置アーキテクチャが異なるプラットフォーム，サーバへの適応に使われる．

4.　具体的なアーキテクチャの抽出

本研究では，概念的なアーキテクチャに基づき，以上述べた 3 種類のアーキテクチャの抽出を試みる．

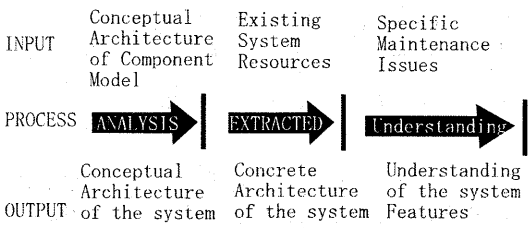

図 4　抽出プロセス

コンポーネントベースの分散アプリケーションを対象とする抽出プロセスを図 4 に示す．

- システムの概念的なアーキテクチャの解析
  コンポーネントベースの 3 階層アーキテクチャに基づいて，システムの概念的なアーキテクチャを定める．
- 機能アーキテクチャの抽出
  対象とするシステムの文書に基づいて，機能アーキテクチャを抽出する．
- 機能写像アーキテクチャの抽出
  概念的なアーキテクチャ，機能アーキテクチャ及びシステムのリソースを合わせて，機能ごとに構成するコンポーネントとそれらコンポーネント間の関係を抽出する．
- 配置アーキテクチャの抽出
  コンポーネントとコンテナの所属関係を抽出する
- 抽出されたアーキテクチャに基づいて，具体的なアプリケーションシステム理解やメンテナンスの問題解決に適用する．

5.　事例研究：E-Process の具体的なアーキテクチャの抽出

E-Process は，ソフトウェアプロセスを支援するためのツールである．Java プラットフォーム上で分散 3 階層システムとして構築されている．E-Process に対して，様々のメンテナンスケースが考えられる．例えば，新たな機能の追加，既存機能のパフォーマンスの改善，セキュリティの強化，実行サーバの変更などに応じたカスタマイズなどである．E-Process のソースコードは，約 10 万ステップで，ファイル数は 775 である．設計ドキュメントも約 200 ページあるが，E-Process をカスタマイズするための情報は不足している．機能アーキテクチャの概要は記述されているが，その機能写像アーキテクチャと配置アーキテクチャがないために，修正箇所の特定，修正に伴う影響範囲について確信をもつことができない．そのため，概念的なアーキテクチャに基づいて 3 種類のアーキテクチャを抽出し，E-Process システムの理解とメンテナンスに適用した．

E-Process[3] システムは，Applet，JSP/Servlet[8]，及び EJB(Enterprise JavaBeans)[12] コンポーネントモデルを用いて実現されているために，それらのモデルから導かれる概念的なアーキテクチャに基づいて E-Process の具体的なアーキテクチャを抽出する．

5.1　Java プラットフォームの 3 階層概念的なアーキテクチャ

3 階層システムの典型的な概念的アーキテクチャは図 5 のようにまとめられる[2]．その構成要素は表示層，ビジネス層及びデータ層である．

(1)　表示層 (Presentation Tier)
　　クライアントとの動的なインターフェースを提供する
(2)　ビジネス層 (Business Tier)

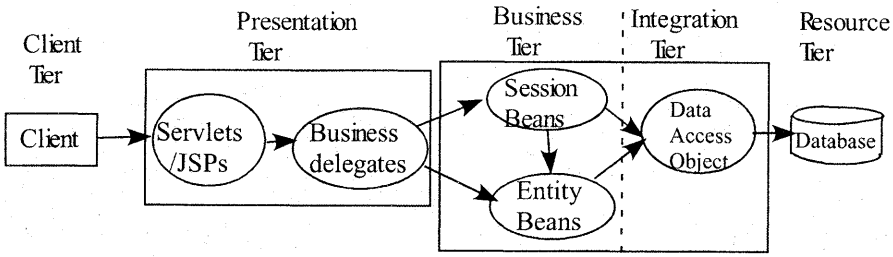

図 5　分散 3 階層アーキテクチャ

ビジネス機能の実行

(3) データ層 (Integration Tier)
DB やレガシーシステムなどの外部ソースへのアクセス

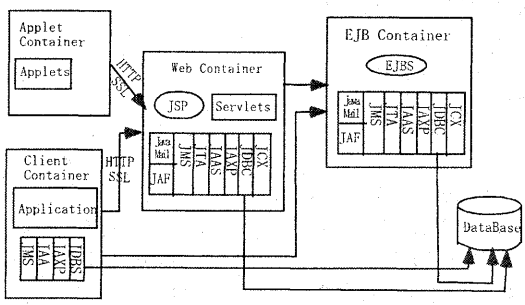

図 6　Java プラットフォームのアーキテクチャ

Java プラットフォームの 3 階層アーキテクチャは**図 6** ようにまとめられる[1]．コンポーネントモデル EJB には 2 種類の Beans がある☆．ビジネスの概念やオブジェクトを具体化し，データの永続性を担当する EntityBean と，Beans のプロセスやワークフローをモデリングする Session-Bean である．それぞれの Bean は外部と交信するために Home インターフェースと Remote インターフェースを用いる．Beans の様々な属性はコンテナへ展開する展開記述子によって記述される．展開記述子がアーカイブ (jar, ear) の Meta-info に含まれる．例えば，JBoss の ejb-jar.xml と jaws.xml により，以下の情報が得られる．

- java，jdbc，sql 間のデータタイプの写像
- EntityBeans と対応するデータベースのテーブル構成と具体的な entity beans をアクセスするための finder の定義

Servlet/JSP は Web コンテナの下で動かされる．JSP は表示ビューをアプリケーションのビジネスロジックから切り離すために導入され，Servlet や Beans で計算した結果を表示する．Servlet の展開記述子 web.xml は web アプリケーションのセキュリティ，Servlet の初期パラメータなどの特性を記述する[13]．

5.2 E-Process のアーキテクチャ抽出

分散コンポーネントモデルでは，各階層の透過性を強調し，アプリケーションが互いの物理的な位置を意識せずに通信できることや，セキュリティメカニズム，プラットフォームの相違をアプリケーションが意識せずにすむように設計できる．

しかし，システムのメンテナンスの立場からシステムを理解する場合には，クライアント階層からデータ階層までのデータの受渡しのためのコンポーネントの接続方法を理解しないと，メンテナンス時の問題の特定ができない．したがって，アプリケーションの階層間の接続と，データ転送プロトコルと，コンテナ管理する内容とを明らかにするためには，具体的なアーキテクチャが必要である．

図 5 の分散 3 階層アーキテクチャと図 6 の Java プラットフォームのアーキテクチャに基づいて，E-Process の 3 種類のアーキテクチャの抽出を試みる．概念的なアーキテクチャの表示層，ビジネスロジック層とデータ層に対応して，E-Process の概念アーキテクチャが**図 7** に示すように構成される☆☆．機能アーキテクチャにより E-Process システムのユーザの役割と機能モジュール，データソースの状況が分かる．

機能写像アーキテクチャに関して，機能モ

☆ EJB2.0 には message-drivebean が導入された

☆☆ 今回は機能アーキテクチャは設計書に詳しく記述されているために抽出を必要としなかった．

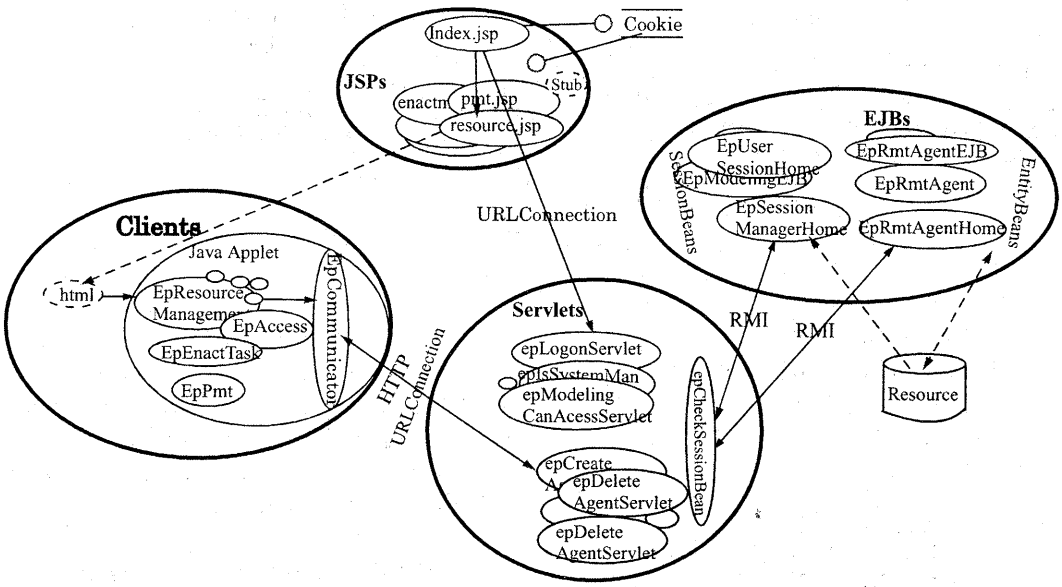

図 8　E-Process 機能写像アーキテクチャの一例

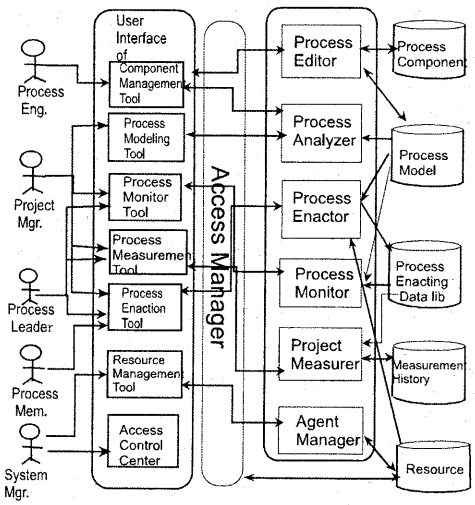

図 7　E-Process 機能アーキテクチャ

ジュールごとに E-Process システムを **Clients**, **JSPs**, **Servlets**, **EJBs** にグループ化し，コンポーネント間の通信プロトコルの検出により，**図 8** に示すような機能写像アーキテクチャが構築できる．図 8 は E-Process の機能モジュールリソース管理ツール (Resource Management Tool) の機能写像アーキテクチャだけを示している．**Clients** と **JSPs** は表示の役割をしている．**Servlets** は機能対応のビジネスロジックを記述し，**EJBs** は関連する Beans を示す．E-Process のクライアント側と Servlet 側，Servlet 側とサーバ側間で明示に使われているプロトコル (HTTP の URLconnetion,RMI，JDBC) を明らかにする．機能写像アーキテクチャにより，ビジネスロジックを実現するための，Servlet で実現した部分と SessionBean で実現した部分との切り分け状況が明確になる．

E-Process は，Tomcat☆ と JBoss☆☆[5] のコンテナの元で動いている．そのコンテナの E-Process 配置アーキテクチャを**図 9** に抽出した．図 8 に示すように機能写像アーキテクチャの **Clients** と **JSPs** は，Tomcat の Web コンテナに収められている．それらはまた表示機能のパッケージと，外部と通信するためのパッケージに分類できる．**EJBs** と **Servlets** のコンポーネントが JBoss の EJB コンテナに集められている．クライアント側とサーバ側との共通オブジェクト csinterface.jar がそれぞれの lib に入っている．配置アーキテクチャを明かにすることにより，異なるベンターのサーバへの移動がしやすくなる．

☆ Servlet サーバ
☆☆ OpenSource の EJB サーバ

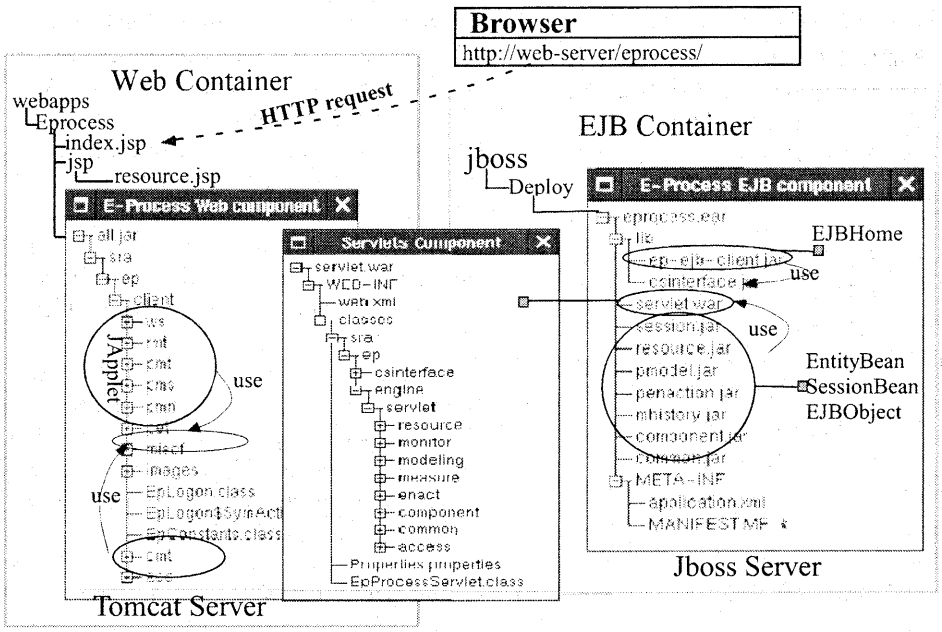

図 9　E-Process 配置アーキテクチャ

5.3 評　　価

E-Process の機能アーキテクチャによって，誰がどのような機能を使い，データがどこへ流れて行くかが分かる．しかし，機能はどのように分割され，どのように分散されているかは分からない．機能写像アーキテクチャと配置アーキテクチャの抽出により，E-Process について，以下の情報が分かりやすくなった．

(1)　E-Process はどのような機能を有するか
(2)　各々の機能がどのように分散実装されているか
(3)　各階階層間のデータがどう流れるか
(4)　システムが故障したときに問題がどこにあるか

そのために，次のような効果が観察できた．

- E-Process のメンテナンスが安定的に行えるようになった．
- カスタマイズやバグ対応する際に，関連リソースを容易に取得できる．
  例えば，システムのユーザ登録機能に関する修正をするときに，機能写像アーキテクチャから**図 10** に示すように階層別のリソースが容易に取得できる．
- E-Process ユーザが既存のサーバに合わせリソースの構成を変える必要があるときに，配置アーキテクチャを使って容易にできる．
- 具体的なアーキテクチャが明らかにしているために，メンテナンスする時の影響範囲の特定をしやすくなる．
- 無効部分の発見
  具体的なアーキテクチャの抽出によって，E-Process システムに使用されていないリソースを有効に発見できる．
- メンテナンス要員の育成
  一般的には，アプリケーションシステムの基盤部分を全て理解した上でメンテナンスに臨むことが難しいために，ここで示したような具体的なアーキテクチャを用いて，メンテナンス要員を育成できる．

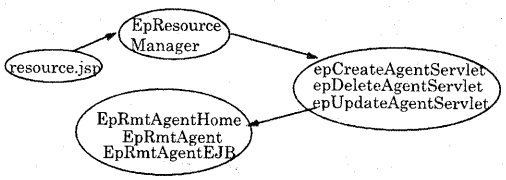

図 10　使用例

6. ま と め

本稿では，分散アプリケーションシステムを理

解するためのアーキテクチャとして，機能アーキテクチャ，機能写像アーキテクチャ及び配置アーキテクチャが必要であることを述べた．概念的なアーキテクチャに基づき，具体的なアプリケーションシステムのアーキテクチャの抽出を行うことによって，その有効性を確かめた．

コンポーネントベースの分散アプリケーションシステムを理解するためには，対象システムのソースレベル解析だけは不十分である．基盤システムを理解した上に分散アプリケーションシステムを理解する必要がある．概念的なアーキテクチャに基づき，基盤システムを考慮しつつ，機能アーキテクチャ，機能写像アーキテクチャ及び配置アーキテクチャにより，アプリケーションシステムの機能構成，制約，依存関係及びインターフェースを明かにし，さらに，システムの適応可能性，変更の影響範囲，構成状況及び拡張可能性について容易に判断できるようなった．

謝辞 御指導と議論して頂いているSRA先端技術研究所の塩谷氏に，東京大学大学院総合文化研究科のKTYYゼミの皆様に謹んで感謝の意を表する．

参 考 文 献

1) D. Alur, J. Crupi, and D. MALKS. *J2EE PATTERNS.* Prentice Hall PTR, Upper Saddle River,NJ 07458, sun microsystems press edition, 2001.
2) K. C.Wallnau, S. A. Hissam, and R. C. Seacord. *Building Systems from Commercial Components.* Addison-Wesley, carnegie mellon software engineering institute edition, July 2001.
3) E-Process Team. *E-Process* **設計書**. SRA and ASTI, 2001.
4) M. W. Godfrey and E. H. S. Lee. Secrets from the monster:extracting mozilla's software architecture. In *Proceedings of the Second Intl. Symposium on Constructing Software Engineering Tools*, pages 15–23, Limerick,Ireland, June 2000.
5) JBoss.org. *JBoss.* http://www.jboss.org/, 1999-2002.
6) R. Kazman and S. Carrière. View extraction and view fusion in architectural understanding. In *Fifth International Confernece on Software Reuse*, pages 290–299, 1998.
7) M. Lehman and L. Belady. *Program Evolution: Process of Software Change.* Academic Press,Inc., 1985.
8) S. MicroSystems. *Servlet Specification.* http://java.sun.com/j2ee/, 2001.
9) H. Müller, O. Mehmet, S. Tilley, and J. Uhl. A reverse engineering approach to system identification. *Journal of Software Maintenance: Reserch and Practive*, 5(4):181–204, Dec. 1993.
10) G. Murphy, D. Notkin, and K. Sullivan. Software reflexion models: Bridging the gap between source and high-level models. In *Proceedings of the Third ACM SIGSOFT Symposium on the Foundations of Software Engineering*, pages 18–28, 1995.
11) H. Sneed. Architecture and functions of a commercial software reengineering workbench. In *Proceedings of the Second Euromicro Conference on Maintenace and Reengineering.*, pages 2–10, Florence, Italy, March 1998.
12) Sun MicroSystems. *EJB Specification.* http://java.sun.com/j2ee/, 2001.
13) Sun MicroSystems. *J2EE: DTDs.* http://java.sun.com/dtd/index.html, 2001.
14) I.T.Bowman, R.C.Holt, and N.V.Brewster. Linux as a case study:its extracted software architecture. In *Proceedings of the ICSE 99*, pages 18–28, Los Angeles, May 1999.
15) K. Wong, S. Tilley, and H. M. Müller. Programmable reverse engineering. *International Journal of Software Engineering and Knowledge Engineering*, 4(4):501–520, Dec. 1994.